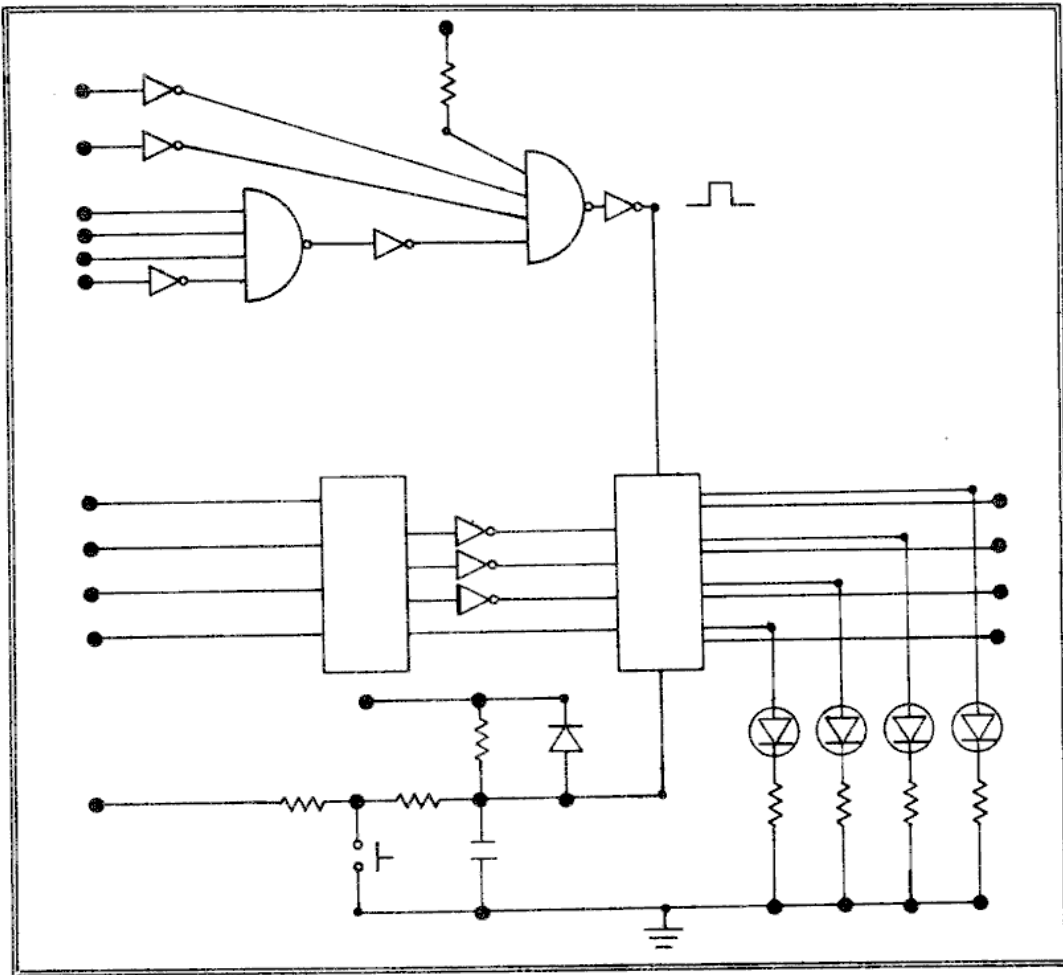


SyncWare News



Volume 1
June 1983 — June 1984

ACKNOWLEDGEMENT

To our wives and children for their understanding,
to our readers for their patience and quest for
knowledge, to our writers for their giving and
loyalty to the cause and to Sir Clive Sinclair, for
his aspirations and foresight, without which...

Table of Contents

SyncWare News Story	1
From The Editor	3
Good News!!	4
Meet the Survivors	5
Products on Parade	7
Cassette Connection.....	18
Forth Language.....	24
LPRINT Hints	28
Byte Pynchers	38
Nim Game	38
Numbers are No-No's	39
Flags, Strings & Other Things.....	40
Hardware Projects	42
Software Compatible Joystick	42
Custom TS	43
Plug Guard	44
Custom Cooler	45
Battery Back-up	45
El Cheapo BBU	45
Rechargeable BBU	46
Built In BBU	48
Uninterruptable Power Supply ..	49
Improved Joystick.....	52
Introduction to I/O	53
Votem Cassette Controller.....	53
Paging Control	56
Veni, Vidi, Video, Vici.....	58
Video Primer	59
Curing the Fuzzies	60
Video Reverse Board	61
VDAQ - A Data Acquisition Program ..	63
Tyd*Byts	73
Listing Scanner.....	73
Hot Z Fix	73
Running on 64K	73
Change RAMTOP without NEW.....	74

Algebra.....	74
Polar/Rectangular Conversion.....	76
Straight Line Subroutine.....	76
Greyplot Dual Display Technique.....	77
Math Development Program	78
Simultaneous What?.....	78
Solving Linear Simultaneous Equations ..	80
Determinants	81
Gauss Elimination	81
Cramer's Rule	82
Gauss Seidel	83
Math Development (SE) Listing, Part 1 ..	83
Using SE in Electronics	89
Wheatstone Bridge	89
Math Development program, Part 2.....	92
Interpolation	92
LaGrange Method	92
Least Squares Method	93
Polynomial Interpolation/Regression ..	95
SE Listing, Part 2	96
Math Development Program, Part 3.....	99
Integration	99
Simpson's Rule	99
Romberg Integration.....	99
Newton-Cotes Formulas for	100
Interpolating Polynomials	
Gauss Quadrature	100
SE Listing, Part 3.....	101
How to Use SE.....	103
SE in Action	104
Math Development Program, Part 4.....	105
Differentiation	105
Horner's Rule	107
Differentiation Demonstration Listing ..	107
Introduction to Integration	108
Arbitrary Functions	109

There are several listings in this book. Since we understand that not everyone is enthusiastic about typing in page after page of program listing, if you would like, you can order any or all of the programs in this book. They are available on four individual tapes. The first tape that you order, comes to you for just \$12.95. Two tapes cost \$20.95; three cost \$26.95 and all four come to you for \$30.95!! Please add \$.50 per tape for postage. Overseas subscribers please add an extra \$2.00 for speedy service.

#1 LPRINT HINTS

#2 MATH DEVELOPMENT PROGRAM

#3 VDAQ

#4 Everything else

For specific concerns or questions about some of the hardware aspects of the programs or of the projects in this book, address your questions directly to Fred. He is our "in house" expert.

The tapes may be purchased from:

SyncWare Co.
902 Hoover St.
Nelson, BC Canada V1L 4X6

ATTN: Fred Nachbaur

The SyncWare News Story

Fred Nachbaur
November, 1984

Three years ago, I received my first ZX81 computer kit in the mail. Only three short years ago, I considered myself a computer illiterate. Since then, things have changed somewhat; but before I tell you about that, let me backtrack a little. I hope that in doing so I might help encourage some of you who are just getting started, and are finding the going perhaps a little rougher than you anticipated. In the process, I'll tell you how SyncWare News came about, and how it is that you are now holding this re-compilation of the first volume.

Before I got my first ZX81, I was more or less familiar with "conventional" electronics, but I found that somehow the computer revolution had passed me by. I'd had a little exposure to the big mainframe on campus during my college days, but that was already a vague memory of stacks of cards with holes in them, big noisy line printers and large white air-conditioned rooms full of stuff that was accessible only to the "high priests" of the computer staff. But that's about as far as it went. I was growing increasingly restive, feeling that I was missing out on the most significant and exciting aspect of contemporary life.

Even just three years ago, the cheapest, most basic computer couldn't be had for less than \$500; so imagine my delight when I saw an ad in "Popular Science" for a computer kit with a paltry \$100 price tag! Well, I strained the family budget and sent my check to an outfit called "Sinclair Research, Ltd." in New Hampshire, hoping that this wouldn't be just another mail order rip-off. A couple weeks later, I got a plain box in the mail; when I opened it, my fears intensified. Oh, dear! Three big chips, two little ones, and a few bags of miscellaneous resistors, diodes and some other stuff. Besides that, there was a small circuit board and a little black ABS box with a membrane keypad to put it all into. "This can't possibly be a computer!" I thought. I assembled it anyway, curious now as to what kind of comic-book novelty I had let myself in for this time. I put it together and (you guessed it) it didn't work. Some digging, through the thankfully included schematic, showed a resistor that wasn't even mentioned in the mimeo'd assembly "additions" sheet. It still didn't work. "Oops, that solder splash shouldn't be there... probably fried the silly thing." Lo and behold, there was the inverse "K" in the lower left corner, just as it was supposed to be!

In the months that followed, I became hooked. It wasn't long before the built-in 1K of RAM simply wasn't enough and I got a 16K RAM pack. At the same time, problems and drawbacks seemed to occur faster than I could get around them. "Slow" mode was excruciatingly slow, "Fast" mode clipped right along but sure messed up the display. The RAMpack would wiggle and I'd be left with a screen of garbage, or the power plug would glitch out at the slightest provocation. I'm sure that I don't have to elaborate further; you're probably tired of hearing these and other horror stories by now.

Yet, these very drawbacks turned out to be a strong point in the machine's favor; they forced me to become intimately familiar with the hardware of my machine as well as with its software aspects. Unfortunately, when I first started there was little "outside" help. SYNC magazine was almost impossible to find. Syntax helped, but it was only one source. So most of what I learned was by direct experience and LOTS of experimentation. After I'd had the machine for about a year, support was growing; but by this time I found that I had learned more on my own than I was picking up from the magazines.

Throughout all this, the traditional computer community disdainfully ignored the machine. It was simply not, in their eyes, a "real" computer, and it almost seemed that the computer shops and magazines went out of their way to put it down (no profit margin, they say). At best, you'd be viewed as an eccentric if you admitted using a Sinclair, and more often than not, you'd get thinly veiled taunts about crashes, overheating and its "ridiculous" way of handling the screen. There was no help there, either.

Then the good news arrived that Timex would be marketing the machine in North America, and the aftermarket exploded in a proliferation of add-ons, programs and other support. The golden days had arrived; the ugly duckling had made good. Unfortunately, the best laid plans can (and do) go awry. Timex's biggest mistake, in my view, was entering the market with a virtually unchanged machine, keeping the membrane keyboard and all the other faults of the ZX81. Meanwhile, they worked on the TS1500, a significant improvement with moving, full-size (Spectrum) keys, built-in 16K RAM, and internal decoding for additional memory, but it was too little and too late! By the time they released the 1500, they had flooded the market with 1000's and acquired a bad name, because they had been selling an experimenter's machine to a dime-store market.

All was still not lost. The damage to the Timex reputation could have been recouped with the TS2068. If only it didn't have those bugs in the EXROM, which prohibited further expansion until corrected. And correction would have simply been too expensive. Speculation? Perhaps so, but Commodore is still doing fine with its C64, which is in many ways an inferior machine to the TS2068. Sure, the home computer market is "volatile," but that alone doesn't make it non-viable.

Please, excuse my digressing. Even before Timex bowed out, I saw a market for a publication for others like myself who needed support for the machine that they simply couldn't get from the available magazines; something a little "meatier," more oriented toward technical applications, and at the same time more down-to-earth about its drawbacks as well as its potentials. Enter, SyncWare News. I had been supplying battery back-up supplies for the ZX81, and had a list of about 200 names of customers and inquiries, and acquired a similar list from another starving supplier. So, I put together some of my own discoveries and programs, and sent out a promotional issue to these lists.

Shortly after the promotional issue went out, I received a letter from a Thomas Bent in Maryland, who asked if I'd be interested in some math routines. "Sure," I said, "send me what you've got." Well, that left it wide open. A few weeks later, Tom sent a very comprehensive well written article on solving simultaneous equations, complete with annotated listing; the works! Not only that, he had plenty more up his sleeve. I decided this fellow was definitely someone to include in "my" endeavor, and made him "Associate Editor." Meanwhile, I kept thinking of other things to write up, Tom's articles kept getting longer, and other authors were starting to take an interest in submitting their input. By the fourth issue, the amount of material was well over twice the originally intended maximum, and even with most of the stuff at 12 CPI and compressed line feed, issue 1:4 filled a cram-packed 48 pages. Meanwhile, subscriptions had grown, mostly by word of mouth, but weren't enough to keep us going. Something had to give.

On a long shot, I approached another fellow who seemed to share the same philosophy of support and open communication, yet who seemed to have a knack for promotion and organization that we technical scatter-brains lacked. That fellow, of course, is Thomas B. Woods, from the same neck of the forest as my beloved computer was originally distributed. He agreed to take over as publisher, and Tom Bent accepted the job of Editor-in-Chief, leaving me free to recuperate from a year to doing most of it myself and relocate to a saner environment. Meanwhile, several outstanding authors agreed to help out and supply their own expertise and support to our project.

The result is SyncWare News, Volume 2, but with the new life brought into the publication by all involved, we soon ran out of the few issues I had left of Volume 1. Rather than simply reprint the somewhat crude dot-matrix originals, we decided to re-edit, reformat, and re-organize the material into a single full size daisy-wheeled journal. You are looking at the result of this project. The material is as current as ever; "The Custom TS" will (I hope) help out as long as there are ZX81/TS1000's to be upgraded. LPRINT HINTS should aid in getting you and your dot-matrix printer on friendly terms, and "Cassette Connection" and the video information will

hopefully make for peaceful relations with your most important I/O devices. Tom's chapter on mathematics is easily the most timeless of all; no matter what machine you have, this material gives more real, usable math information than stacks of conventional textbooks. It is our sincere hope that SWN Vol.1 is one publication you will refer to again and again as you make your way from "computer illiterate" to "certified hacker."

Much more is in store for Volume 2. As mentioned, other authors are sending significant contributions, and SWN is no longer just a two-man show. We've expanded to the new machines (TS2068, Spectrum, QL [when it arrives], possibly Memotech MTX512 if the interest is there). We intend to continue the "Spirit of the ZX" by providing hardware projects, machine-code programming, alternate languages, and technically oriented applications. Emphasis will remain on material of a more advanced nature, but we won't leave the beginner behind. In my first editorial I wrote the following:

"New users will also benefit from SWN, even though we will assume that you have a working knowledge of the machine's basic operation. Even concepts that are perhaps over your head at first will become clearer as you gain proficiency, and your collection of SWN will be a valuable resource as you progress beyond beginner-level books and other publications."

"Don't forget!! Even the most incredible program was written by someone who was once a beginner, mystified and overwhelmed by the apparent endlessness of it all. Our writers will do their best to make your education as painless as possible, by discussing the 'why' as well as the 'how.'"

Most probably, there is no end to it all. Once you're comfortable with the present state of the art, it will not be too difficult to keep up with new developments. I sincerely hope that SWN 1 will continue to help you catch up, while SWN 2 (and 3, 4, etc.) will help keep you abreast of changes. So on that note, I'll end off and wish you all continued "Happy and fulfilling Sinclair Computing!"

From the Editor...

Thomas Bent
Editor

As with Fred, my entry into computers came about three years ago, but my reasons were slightly different. A change of jobs brought me into the computer era. Several college courses in programming and numerical methods helped bring me up to speed. My ZX81 gave me a real edge over my classmates though. Not only did I practice my school work at home, but I became proficient in converting between languages (Basic, Fortran and Pascal).

Enter, SyncWare News. I received Fred's Vol 1/1 issue and got quite a kick out of seeing the math applications that he had developed. I sent my inquiry/reply to him (tore off the back cover) and found his return response soon afterward. Well, that started it (much to the chagrin of my wife).

When Fred relegated responsibilities to Tom Woods and me, the SWN outlook improved rapidly. Tom's tremendous subscription drive, the letters we received asking us to continue and several endearing friends to all of us, who have volunteered their support, have insured SWN's survival. This volume is a tribute to all of you.

The first thing Tom Woods wanted to do, besides upgrade, was reprint back issues. There were no masters. This is a complete recompilation. Fred started recompiling in late September, I started editing in late October and Tom Woods started strapping it all together in late November. There were only minor annoyances to get in the way, such as, moving, SWN21, SWN22, SWN23, Profile 2068, Updates and working for a living.

The computer technology has advanced at just an incredible rate, even for our forsaken Sinclair computers. As we (everyone) learn more about our machines, so do we learn about ways to improve what

has come before. In this rework, I amassed an incredible pile of cassette tapes. Eighteen months of work spewn here and there, and all the while trying to get organized. From the Cave-man word processor to the cassette based Memotext, to a high speed Memotext and finally, as I write this last editorial, a Compusa disk based Memotext, with Memotext relocated above 40K! It's just incredible!

This volume will be valuable to all who TAKE the TIME to read or even study its contents. You will find an incredible wealth of information herein.

Again, through technology, The amount of text compression on these pages will astound you. Proportional spacing can really cram information. Each page in this book contains about 3 pages of normal size text. The revised schematics herein were enlarged to make them clearer and prevent them from being "lost in the corner." I took the liberty to denote connections by large circles. I hope you find them legible.

Some of the material here (especially the math) is even more applicable to the 2068. After experimenting on your 1000 and gaining confidence, (you may find tinkering easier and more fun than you thought), making some 2068 modifications may be easier to justify. There will be a lot more for both machines in the years to come.

GO SLOW! Enjoy it, and may the Power be in your hands!

Make no mistake! This ENTIRE book was written on ZX81's (32 column screen and inverse capitals), Memotext with 64K Ram and Memotech I/F and printed on a G. Itoh Starwriter F-10. However, the annotations for the SE program were written on a 2068 with Mscript and a Tasman I/F.

GOOD NEWS!

GOOD NEWS: "THE NEWS" LIVES!

Shortly after SWN 1:4 was mailed, letters started coming in from concerned readers, encouraging us to continue publishing. It is largely thanks to those individuals that I can make the following announcement: SyncWare News will continue into its second volume. We have been convinced that there is still plenty of interest in the TS machines, and we've decided to keep giving the best we've got in support.

I suppose that in itself would be good enough "news" for most of you. There's much more, though! I am pleased to announce that Mr. Thomas B. Woods is now part of our little group, and has consented to take over in the capacity of Publisher/Promotion Director. Most of you have dealt with Mr. Woods, or at least heard good things about him and his "ZX PRO/FILE" package; you know that he is as dedicated to our machines as anyone can be. More than that, he has a better business "head" than yours truly, and has determined to help make SWN financially viable. He also has an impeccable eye for the graphic arts, and as a result SWN Vol. 2 will have a "new look." We are going to a full-size format (but will still be reducing text to bring you the same content). Don't worry about readability; this issue is the last you'll see of the dot-matrix output from our trusty Gemini-10's. In volume 2, all text will be produced on a professional daisy-wheel "Spinwriter" printer (but still driven by a ZX81 and Memotext or TS2068/MSCRIPT, of course!) Add to this, increased attention to artwork, and we're sure you'll find SWN 2 as appealing as any dedicated magazine you've seen.

Tom Bent, who you all know and love by now, will take over as Managing Editor, and SWN editorial offices will be moved to his address. It will be his job to edit, arrange, print-out, and paste up the issues, and generally get them "camera-ready" for Tom Woods to print and distribute. Meanwhile, I will keep doing whatever it is that I do in my capacity of Story Compiler/Research Director. I'll still be working closely with Mr. Bent in getting together the articles and programs in the magazine. With this division of labor it should be possible for us to "clean up our act" as regards publication frequency; no more "double issues" (as 1:4) or unduly late issues, once the initial confusion has blown off. Publication will remain bi-monthly, starting with the September/October issue. Each issue will be 16 full-size pages, packed with the kind of articles and tutorials you're used to from us by now.

To make this all possible, we have to increase the subscription rate to US\$16.95 (North America; US\$22.60 foreign) for Volume 2 (a mere 13% increase over volume 1). We are presently accepting orders (check, MO, VISA, or MC). We are also accepting paid advertising at reasonable rates; if you look over this issue you'll see the first of the ads "to get the ball rolling."

Here's who to send what to. Send subscription orders, advertising orders and queries, and related matters (especially money) to Publisher, SyncWare News, c/o Thomas B. Woods, PO Box 64, Jefferson, NH

03583. Also send comments on format, etc., change of address notices, and such to Mr. Woods, or call him at (603) 586-7734.

Send "Letters to the Editor," suggestions for content, program/article submissions, and other things you'd normally send to an Editor, to Thomas Bent, 9016 Flicker Place, Columbia, MD 21045. Any of this sort of material may alternately be sent to me, Fred Nachbaur, 902 Hoover St, Nelson, BC V1L 4X6, Canada, since as I mentioned Tom and I will continue to work closely together to get the "goodies" into your anxious hands. So send your comments, etc., to whichever of us you feel more comfortable dealing with.

Now, a look at what we've got planned for the next few issues. If you're a John Olinger fan, go out and celebrate; John's long-awaited TMS9918A Video Upgrade project will appear in a two-part article in 2:1 and 2:2. In 2:3, we'll publish his 2764 EPROM cartridge board project; this will let you have your most-used 16K programs on-line about as fast as you can enter RAND USR.... If you want a "jump-start" on these, drop John a line and request his product sheets; you'll be amazed at the contributions this man has made to your machine (incl. TS2068).

Speaking of 2068, here is a beautiful machine that "died on the vine," as it were, but from the mail we've been getting, it seems that a lot of folks have made Timex's loss their gain, and now own one of these gems but don't know where to turn for support. So we will be paying more attention to this machine in SWN Vol. 2. Where's a good place to start? Well, how about the ROM / operating system; the information about the "ins and outs" of this machine is sparse indeed. Now who in the world do you suppose could write about this? The requirements are that the author of such articles be intimately familiar with the unit, knowledgeable about computing in general, and an excellent writer. There's only one person who really "indicated" to me, and that's Ray Kingsley. After all, anyone who can rewrite HOT Z for the TS2068 has GOT to know what goes on inside that pretty silver case. So I approached Ray with the idea, and surprise and delight, he answered, "Well, ok..." The first article of the series will deal with the jump tables, subsequent installments will discuss the SAVE/LOAD system, the function dispatcher, and related topics.

Another author who has tentatively agreed to help fill our pages is Gary Smith, of Hawg Wild Software, who (if response is favorable) will grace us with a tutorial series on the FORTH language. The introduction to this series is later this issue; please read this and the editor's note ASAP.

Many of you know of Bob Berch (Cinagro Software), who has provided a great BASIC compiler and other utilities for the ZX81/TS1000 machines. He has agreed to let us print his nifty Serial Printer Port project. What's neat about this one is that 1) it takes very little additional hardware (a couple op-amps) and 2) it doesn't require connection to the rear edge-connector, obtaining the data from the MIC jack (!) instead. Talk about elegant! In addition,

Peter Hoffman of Delphic Enterprises is planning an article on writing relocatable Z80 machine code (applicable to all Timex/Sinclair machines, as all have a Z80 at the heart). Paul Hunter has indicated that he may send us further articles on building control hardware (again, assuming that the interest is there and you all let him know that you want it). Paul Donnelly (see ENER-Z review this issue) will show you how to use an output port board such as the ENER-Z or the Byte-Back BB-1 to control a 3-digit, 8-segment LED display for continuous data monitoring. I'll continue my ramblings on various things, though I hope that having someone else edit it will reduce my verbosity. In between his other duties, Tom Bent will keep you abreast of math developments. Maybe we can even talk Tom Woods into throwing in a occasional article when he can find the time.

We're all very excited about Volume 2, and we're sure you'll be too. But don't forget; it all depends on you. Right now we need subscription and advertising orders; so send your check/MO to Mr. Woods TODAY - correction, RIGHT NOW! Meanwhile, enjoy this issue. Regards,

Fred

(Publisher Pro Tem)

Meet the Survivors

Well, now you know that SWN is among the publications that survived the Timex "crash." Thank God that we're not the only one, though; we are extremely fortunate to have several respectable publications that still support "our" machines. But first, who didn't make it? At the top of the list, of course, is SYNC Magazine. Ziff-Davis is trying to get you to switch your subscriptions to either "Creative Computing" or "Computers and Electronics." I should mention, however, that recent issues of both have been disappointing to Sinclair/Timex users. It seems that ZD has dropped the "line" like a hot potato. If you haven't committed yourself yet, I suggest asking for a refund, since the options they give you don't provide what you paid for, and that is support for your machines. If you have traded your Sync subscription for one of the others, write to the editor of that mag and tell him in no uncertain terms that you WANT ZX/TS support. Another "deadie" is Timex-Sinclair User magazine, though rumors still persist that it is being revived. (I doubt it, myself, and didn't really find it all that great anyway.) TS USER (the newsletter) has also bombed, right after staunch assurances that it would continue. But that's about it; the rest are still around, and determined as ever to keep on going. Following is a summary of some of the still-available publications. Why would I "plug" the "competition?" Quite simply, because we don't view the others as competitors, rather as colleagues; as I said before, no one publication can cover it all, and if you're really serious you should support ALL of the sources listed. By pulling together and supporting each others' efforts we ALL benefit.

- * SYNTAX, The Harvard Group, Boulton Road, Harvard, MA 01451. This is the "Wall Street Journal" of the Sinclair-Timex world. They have been around

since "day one," and in spite of ups and downs over the years still continue to be the best barometer of Sinclair-related goings-on to be found anywhere. If you're not a long-term Syntax subscriber, you should get "The Works" which includes every issue to date, plus the three issues of the ill-fated (unfortunate, as it was excellent) Syntax Quarterly magazine. You'll need these issues of SQ as background to the upcoming John Olinger articles in SWN. Most of the "tricks" and programming tips we now take for granted first appeared in Syntax. Perhaps what I like most about it, though, is the quality of the journalism; you won't find any half-baked rumors or "National Enquirer"-type stuff here. It may seem a little expensive, but when you look at what it must cost to maintain world-wide information sources, etc. in addition to the usual expenses, it starts looking more reasonable.

- * TS HORIZONS, 2002 Summit Street, Portsmouth, OH 45662. This is a relative "newcomer" to our field, appearing just shortly before the Timex withdrawal. I'm glad its publisher, Rick Duncan, decided to keep it going, as it's a nice little mag at a very reasonable \$12/year (12 issues) rate. Of all the survivors, this is the one most oriented toward beginners, though there are also "heavier" articles for the more experienced. TS Horizons has a pleasant, homey "air" about it, and provides a good offset to the rather formal flavor of Syntax.

- * COMPUTER TRADER MAGAZINE, 1704 Sam Drive, Birmingham, AL 35235. [DO NOT confuse "the Trader" with "The Computer Shopper"; not the same thing at all.] This is by far my favorite of the general-interest computing magazines. It is the friendliest, most informative publication available, and one which the flashy but fluffy mags "on the stands" would do well to emulate. It was started just a couple years back by Chet Lambert, and began as a small product flyer. Chet has since nurtured it into a 100+ page magazine, complete with color covers and the other "amenities." Through it all, it has maintained a "grassroots" philosophy and a friendly attitude that doesn't talk down at the reader. Most importantly, for our purposes, is that the magazine dedicates a good deal of space to Sinclair/Timex support. In the current issue (June 1984) there are six dedicated TS articles, as well as ads for aftermarket TS suppliers. Meanwhile, it will expose you to what is going on in the other micro "communities" (Atari, Apple, Commodore, Kaypro, TRS-80). Also, an emphasis on amateur radio makes this a great mag for the "hard-hacker" and the communications experimenter. A real bargain at \$15 for 12 issues.

Besides the mags, there are many informative newsletters available. Most are published by User Groups, and VERY FEW, if any, ceased publication when Timex quit. In fact, there are probably MORE good newsletters available than there were a year ago. Subscribe to several of these; this is the best way I know to find out about new discoveries as they are being made. Some of my favorites are listed below.

* QZX, 2025 O'Donnell Drive, Las Cruces, NM 88001. (The Journal Covering Amateur Radio and Sinclair Computers.) Are you interested in communications using your ZX/TS? Well, QZX will show you how to do it. Morse code, RTTY (radio-teletype), calculations (antennas, filters, you name it), satellite tracking... it's all there. There is of course a QZX net, on which you short-wave listeners might pick up some tips. Then, it won't be long, if you aren't a ham already, before you practice that ole Morse code and bone up for the FCC "ticket" to get on the net yourself. It's custom-made for anyone wanting to communicate on the TS (and on a budget). If you're a ham, and own a ZX/TS, you'll find QZX worth many times its price of \$15 per 12 issues. If you're not a ham, but you like electronics, then you'll still find a lot of "beef" here for your related projects (e.g. video, control, data transmission, etc.)

* TSUG Newsletter, c/o Doug Dewey, 206 James St., Carrboro, NC 27510. From the high-tech "Research Triangle" in North Carolina comes this info-packed general interest User-Group publication. This is almost a model for others; the publication is professional but friendly, and often includes some fascinating surprises (enclosures provided by suppliers, samples of other publications, etc.) The TSUG organization is a "model," also; in the interest of efficiency they have legally acquired Not-for-Profit status, part of the reason they can send you TSUG Newsletter at the low membership fee of \$10/year.

* SYNAPSE, Central PA TS User Group, c/o Bill Russell, RD 1 Box 539, Centre Hall, PA 16828. If you've dealt with G. Russell Electronics, you've probably seen SYNAPSE. It's one of the smaller periodicals, but don't let its small size fool you. These guys manage to cram more real content in 8 half-size pages than some mags put into 200 pages of fluff. Still one of my favorites. Subscription; \$12 per year (12 issues) plus this gets you on GRE's mailing list.

* L.I.S.T., Long Island Sinclair-Timex Group, c/o Paul Donnelly, 10 Idle Day Drive, Centerport NY 11721. This latest addition to the Sinclair-related newsletters has appeared for only a few issues so far, but I must say I'm impressed. Under the leadership of Paul Donnelly, it provides excellent support for the more advanced as well as the beginning user. As an example of the quality of material to expect, see Paul's review of the ENER-Z board in this issue. Another of Paul's articles appears in the next issue; and since in the publishing world "reprinting is the sincerest form of flattery," you should already know what I think of Mr. Donnelly's work. Subscriptions to L.I.S.T. Newsletter is \$12/year and includes membership in the group.

The following publications are either magazines or newsletters, and have been added "after the fact" to this list, to provide even more choices for you. Contact them for more information about their content, etc.-ed.

* CATS, PO Box 725, Bladensburg, MD 20710. \$12.00/yr.

* SUN, 3224 NW 30th Ave., Gainesville, FL 32605. \$12.00/yr.

* BASIC Computer Programs, 3705 Biscayne Blvd., Miami, FL 33137. \$12.95/yr bi-monthly.

* Sinus News, PO Box 523, Owego, NY 13827

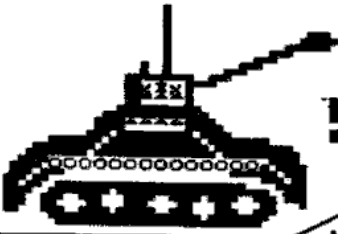
* Time Designs, 29722 Hult Road, Colton, OR 97017. \$3.00 for first issue.

* Timelinez PO Box 1312, Pacifica, CA 94044. \$12.00/yr.

That's about all the space we have this time. We'll continue to keep you informed of other periodicals in future issues; meanwhile, if you have a newsletter, send us a copy for our reports and reviews. By the way, you may notice that my "list of favorites" is very close to the one in a similar article in our pilot issue over a year ago; fancy that! Small world, eh?

ATTENTION TECHNICAL PROGRAMMERS!

If you've written or are writing
 a real "Tank" of a program,



SYNWARE CO.

WANTS YOU!

Y=K+T*(U*SIN
 LET
 E+A*T/2;
 IF Y<0
 THEN
 LET
 Y=0

PRODUCTS ON PARADE

Memotext

Memotech Corp. Fred Nachbaur
99 Cabot Street or 902 Hoover Street
Needham, MA 02194 Nelson, BC CANADA V7L-4X6

This word-processor from MEMOTECH is a tough thing to review, as it would take more space than I've got left to cover all its features. Simply stated, it is a marvellous piece of work. You could compare Memotext (M/T) against commercial word processors for fancier machines costing hundreds of dollars, and the T/S - M/T combo would come out ahead in many cases. If you have a printer and a Memotech I/F, you really should get one. Sell your typewriter if you have to.

One of the nice things is that the keyboard is re-defined so that it's like a typewriter. If you're a typist, then you'll feel pretty much at home. It takes some re-adjustment for "old timers;" it just doesn't "feel" like your good ole ZX81, and you'll tend to stumble around a little when going back to programming after doing a bunch of Memotexting. Another mind-boggler is that if you've been word processing with the Memotech I/F; Memotext also reverses the cases, making inverse video upper-case. But boy, is it quick! You'll find it challenging (impossible?) to out-type. If anything, the debounce is a little too fast; with soft-touch keyboards you might sometimes get double characters if you don't hit the key dead on. There's full auto-repeat on all keys. Error trapping is extensive and almost too foolproof. Files can be saved one at a time or all at once.

What don't I like about it? For one thing, text files can't be split or merged, and when printing a text file paper is automatically advanced to the next page, no matter how short the file. Great for letters, but annoying for newsletters. I should say that there's probably an easy way to get around this using data files (a separate kind of file usually used for inserting "customizers" into form letters, mailing lists, etc.) but I haven't learned near all the tricks yet. Really, my only "gripe" is that M/T takes over completely. there's no way to return to BASIC when it's on. But I suppose that would really be asking too much. [Since this article was first printed in SWN 1:3, we've come up with a way to run M/text in RAM, and have included a QIT function, among other things; contact us about availability. - ed.]

Although M/T is easy to use (with atypically complete documentation - done with M/T of course) it will provide you with a challenge to use it to the fullest. It will take some thought and plenty of experimentation (and paper) to really make it do all you could want. It's well worth the effort. And oh yes, it's certainly worth \$49.95. If you don't do much writing beyond short notes, etc. it might be more than you need; but if you like to write and are frustrated with BASIC WPs, this one's for you.

Key

G. Russell Electronics
RD 1, Box 539
Centre Hall, PA 16828

KEY is a utility of the tool-kit variety, containing "Protect" and "Merge" routines which allow you to load two separate programs and combine them into one listing. Also included is a "bytes remaining" routine, and the star of the show, "Unlock," which lets you break into those un-LIST/SAVEable tapes for modifications and personal back-up purposes. All are easy to use and reliable. The package resides above RAMTOP, and it is thus immune to NEW. Best of all, KEY takes up only about 1/2 K, making it a viable alternative to the larger toolkits when memory is tight. Even if you have other toolkits, this one will make a useful addition to your utilities collection because of its compactness. It comes with a well-written "scoop sheet," and is well worth the \$10 (postpaid) asking price.

Z/XLR-8

G. Russell Electronics
RD1, Box 539
Centre Hall, PA 16828

This (pron. "zee-accelerate") is a "speed load" program which you'll love if you're going grey waiting for those long tapes to load. It takes up 2K of memory, and may be located anywhere in RAM (e.g. the 8-16 block). It is all in software, so there is no additional stuff hanging around your computing space. Z/XLR-8 is very well documented in a 20-page manual and costs only \$11 (cassette) or \$21 (EPROM).

One of its nicest features is that rate of data transfer is set when you initialize; rates from about 4 to 10 times normal speed are possible. Not only that, the cassette package comes with a calibration program to help you determine just how much you can "XLR8" the save/load process before your cassette recorder imposes "hardware limitations." the faster you want it to go, the better your cassette machine (and tape!) must be.

But even that's not all. In addition to program SAVE/LOAD (analogous to normal tape routines), you can also SAVE and LOAD along the screen display, any array, or any selected part of memory (e.g. machine code, etc.). You can also do hi-res graphics save/loads if you have the Advanced Interface Designs graphics board. Even that's not all! You can do an "index load," in which your tape is scanned for the file names and types it contains.

The one thing I didn't like is that there's no way to abort the LOAD functions once started. So if you give it a wrong file name you're sunk and have to start from scratch. This is not a problem with the EPROM, of course, but it can be annoying to have to re-load Z/XLR-8 "the hard way." If you realize you entered a wrong file name before specifying function, you can recover by entering "BS" (Binary Save) and small integers for "start address" and "No. bytes." The computer spits out those few bytes and you're back in business.

Z-XLR8 was originally supplied by Brad Bennett's (the author's) company Advanced Interface Designs, but distribution is now through G. Russell Electronics. Definitely a good product, well worth the price. Since it is very flexible, it takes some experimentation to get the best results with your particular system; but the rewards are great. Wouldn't you love to load a full 16K program in less than a minute? Well, now you can.

HRP

G Russell Electronics
RD 1, Box 539
Centre Hall, PA 16828

HRP (high-resolution printing) allows you to individually address each dot on your TS2040 or ZX printer. It sets up a 256x192 pixel map, taking up about 6K, which you then fill with your data. You can then dump the map to your printer. The program is well-documented with several examples, and the tape even includes a ready-to-run application example - least-squares regression with both low and high-res plotting. The result is quite impressive; if you didn't know better, you might have trouble believing that the output came from a ZX/TS. I like it because it's all in software and is easy to control. Since each dot takes up one bit, the map takes a sizable hunk of memory, but this is not too great a price to pay for the beautiful and relatively fast high-res printouts you can get. Besides, the map is re-locatable beyond the 32K address, if you have a 32-64K RAM. All in all HRP is a quite unique program and well worth checking out.

You probably already know that G. Russell Electronics supplies the popular "Winky Board II" passive cassette filter. You may not know that they also have some fine M/C utilities, at reasonable prices. Bill Russell spear-heads the company, and sent evaluation copies of KEY and HRP and arranged for a sample of Z-XLR-8.

Bill Russell is also the driving force behind the Central Pennsylvania T/S User's Group, a dedicated group of ZXers near Penn State, and the only club I know of that meets at least once a week (they still meet every Wed. night at Penn State-ed). Really devoted ZX hackers! Bill also publishes a small but informative monthly newsletter, SYNAPSE, containing programs, reviews and helpful hints. Subscription = \$12 per year.

Graphics A to Z

Authors: Paul Bingham (Art(s)iste ZXtraordinaire)
Rick Goulian (Low Level Language ZXpert)

Publisher: Pleasantrees Programming, Tucson, Arizona

This long awaited paperback covers something for everyone who may want (or need) to draw or plot to the display file. This reference manual (just chocked full of goodies at 200+ pages), has a program on practically every page. It covers such off the wall things as controlled crashes (good for visuals), to sophisticated multifigure movement of such creatures as ZXak(pac)-man in two dimensions, and touches on the really tough stuff (three dimensional overlay movement for smooth diagonal scrolling). This feat is the basis for their game "City of Xon." Techniques for drawing simple graphic figures to complex math graphics and designs are discussed in detail. ZX listings are provided (proof of the pudding) as well as screen dumps for many of the programs.

The most interesting part of the book (for me anyway) is certainly the several chapters covering the use of the Memotech High Resolution Graphics interface. This part includes not only programs using the various commands, but also an explanation of each of the 31 commands AND a full disassembly of the HRG ROM. Many of the programs come compiler (a la Bob Berch V1.1) ready. This is quite nice as one of the slower features of ZX BASIC is printing to the screen. The speed increase is naturally "breathtaking."

Did you ever envy those machines that can list more than 10,000 lines? Well don't turn green because the solution is in the book. It's a little tedious to say the least, but it does provide a good place to tuck away some often-used routines. How about upper- and lower-case compressed characters? It is right here with much more.

Well, as you can see, ZXcellence, but at a price. \$17.95 brings this no-filler vehicle to your hands. However, you do get what you pay for. Is it for you? If you have HRG and/or a compiler then I recommend this book highly, if not then it is a little over priced. I have the book, but no HRG. I am thinking about getting HRG now that I have more insight into what I can do with it (without all the trial and error). Think of this book as a text, as you will have (want) to study it. Many of the techniques used for entering programs are quite tough, and I suggest a good assembler/ disassembler to help input and debug some of the listings.

Report Generator Board
ENER-Z Company
PO Box 635
Fort Washington, PA 19034

FEATURES:

Real-time clock, A/D, I/O Ports, parallel printer interface

PRICE: US\$ 89.95 A&T, \$59.95 kit

If you could buy only one peripheral which would let your ZX/TS communicate with the real world, Ener-Z's Report Generator just might be it. The board's features include a real-time clock, analog to digital interface, input/output ports, and a Centronics printer interface. All those features, and the under \$90 price tag, sounded like too good a buy to pass up to me, especially since one or two of these features alone would cost more than this board from most other vendors. Does the R.G. deliver? Let's take a look.

Physically, the R.G. board is 4" x 6" and has a cleanly designed layout for its 16 ICs and assorted passive components. Construction is professional, with few jumpers, plated-through holes, and sockets for the more critical ICs. An on-board regulator is supplied; this helps keep your ZX/TS's internal regulator cool. A battery holder is provided for the real-time clock (RTC) and most of the offboard tie-ins can be made using standard DIP (dual in-line pin) headers, which are easy to obtain and inexpensive. In most uses, extra ground or 0 volts lines are required. These fall between the active lines and help keep down the effect of transients (glitches) in your ribbon cables to the real world. The board plugs onto the ZX/TS expansion port and a "through" edge connector is provided for other add-ons (e.g. RAMpacks).

The heart of the R.G. board is a Z80 PIO (parallel input/output) chip. Through clever use of an internal data bus (on the R.G. board) Ener-Z has expanded the capabilities of this simple chip to allow it to address what, in effect, are more than five separate port configurations instead of the usual two. Two versions of the R.G. are available, one has all its software on a 2716 EPROM (mine is this type,) the other has the driver software in RAM. The RAM version is required if you already are making use of the 2800-2B00h area where the EPROM would normally reside. All functions on the R.G. board are accessed with USR calls to the appropriate subroutines on the EPROM, which in turn executes the I/O functions.

An OKI MSM-58321 clock chip provides the year, month, day, week, hour, minutes and seconds for the RTC. With 3 Nicad batteries installed and the clock setting software (supplied as a listing) entered, you need only enter the correct time once and your ZX/TS will always know exactly what time it is. This is especially helpful if you want your computer to perform certain I/O functions at specified times.

Eight-bit input and output ports are provided by a 74LS244 buffer and 74LS373 latch, respectively. The inputs can monitor digital signals in the TTL range (0 or 5 volts.) Real-world signals can either be directly coupled to the buffer lines (e.g. simple

switches) or coupled through conditioning circuits (e.g. relays, optocouplers.) Outputs are TTL compatible, although load-carrying capacity is only moderate. To drive a large fanout or heavy load, you'd need power amplification/ buffering stages (e.g. optocouplers + triacs to run 110 VAC circuits.)

The analog converter section of the board uses a National ADC0809, eight-bit port, eight-channel input, to convert real-world signals into their digital equivalents. As an example, a measurement of 50 means that the ADC input is seeing about 1 Volt DC. With 8 channels, you can monitor the status of 8 separate analog devices. Typical uses here include monitoring proportional-type joysticks (paddles,) thermistors, photo-voltaic cells, peak-detector circuits, and pressure transducers.

Up to this point we've seen how this board can tell you what is happening off-board (digital input and A/D), know when it's happening (RTC) and even perhaps do something about what it sees (output). Now we're at the stage where, perhaps, you can see how the board got its name. The unit can also generate physical reports about what's going on, this time by using its final feature, a Centronics parallel printer interface. For this important, high-speed task, the PIO ports are used directly. Eight data lines transmit ASCII character codes generated by a look-up table in the EPROM. The EPROM effectively "overlays" the Sinclair ROM for this operation, allowing the support of Sinclair's LPRINT, COPY, and LLIST commands directly, for the printing of the standard character set. Graphics are not directly supported, but special USR calls can produce a byte code which will produce other character depending on your printer's capabilities. The overlay of the Sinclair's ROM is done by overriding the ROMCS NOT signal when the printer commands are invoked.

The other two lines provided by the printer are STB (strobe) NOT and ACK (acknowledge) NOT. I used the interface with a Centronics 101 printer and Ener-Z says it works with Epson and should work with most other Centronics-standard printers. I agree, but suggest you include your printer type in any correspondence with Ener-Z.

After getting my R.G. board, I put it through its paces, on and off, for over a month. The RTC lost only a few seconds, and I hooked up switches (joystick, actually) to the input ports. I've used the output ports, all eight lines, with two off-board chips to drive a 3 digit, 7-segment LED display [Reported in Vol. 2/2 -ed.] This setup lets me see measurement status without turning on the TV. The A/D converter measures some resistance circuits I've set up quite accurately, and I've even tied it into a thermistor and gotten a fairly good thermometer, accurate without amplification to about 2-3 degrees F. To get the printer interface working, you'll need to make up your own cable. You can get everything you need at Radio Shack or a number of the mail order houses. Try to keep your cable length under 8-10 feet in length.

If the R.G. board sounds pretty good up to this point, make no mistake, it is. However, it seems that nothing is ever absolutely perfect and there are some negative aspects to the board. Minor turn-offs which don't really affect the board's performance included a loose battery connector on my board, some "afterthought" pull-up resistors tacked on to ensure TTL-CMOS compatibility, and the lack of a case. Three more meaningful items, which might even affect your decision to get this R.G. board, do need mentioning. First is the somewhat hastily prepared documentation. While there is an excellently documented source listing for the EPROM, the supporting narrative on BASIC applications has quite a few typos and no page numbers. The 29 page documentation swings back and forth between the EPROM and RAM-based USR calls, and while a mini-hex monitor for loading the RAM version (you have to enter this by hand) is provided, the hexcode listing is not. I'd suggest that Ener-Z either prepare two separate versions of the documentation or one complete and comprehensive dual version. I think the average user will be able to use the board, but straight BASIC programmers (i.e. who don't use ML and have little hardware knowledge) may find they'll have to make a call or two to Ener-Z for guidance. I have found them most helpful on the phone.

The next complaint is that we're not told explicitly in either the advertising or the documentation, which port locations are used. In the case of the R.G. board, with a little digging, the source listing does indicate addresses 01, 03, 05 and 07. However, a quick look at the board's partial decoding will show you that ANY odd numbered port below 80h will respond to the I/O calls from EPROM. Keep this in mind if you have other I/O mapped peripherals.

The other significant "flaw" with the R.G. board has to do more with the "nature of the (Timex/Sinclair) beast" than with the board itself. As you probably know, the ZX/TS cannot directly address I/O ports. In order to do so, then, we just jump to a machine language routine in FAST mode, and consequently lots of blank screen time and "flicker." I probably wouldn't have thought this to be a significant problem if I hadn't seen JK Audio's 310 board. While this board costs about as much as the R.G. and has fewer features (I/O ports and clock only,) it uses memory mapping to accomplish its I/O function. I don't feel the R.G. board should be downrated on function because of the difference, only that users should be aware of a need for a little more effort required to produce good-looking screens with the R.G. board.

To more than make up for these little flaws, the R.G. offers its high reliability, numerous features, low cost and some bonus features not mentioned in its documentation. The board nominally has one 8-bit input and one 8-bit output port. You could conceivably stretch these, with some off-board multiplexing, to 64 each. How? Schmitt (or comparator) conditioning of signals fed to the A/D converter could give you 8 more digital input lines. Also, the printer port has 8 bits, this time with handshaking. There are other such combinations which you could implement with a few ICs to suit your particular application. A final extra bonus is that about 180 bytes of the EPROM are not yet programmed. Using sufficient care, and an EPROM programmer, you could use this area to add your own special USR routines to the board.

On balance, I found the R.G. board an excellent value and perhaps (next to my \$39.95 Timex) the most cost-effective computer purchase I've made. If you need any two of the board's features, I recommend buying it. I've given this board a hard-to-get 9.9 on my price-for-value scale. A "10" could be obtained if the RAM-driven version were provided on tape, and documentation cleaned up a little. Ener-Z has done an outstanding job with its first entry into the ZX/TS peripherals market. As a suggestion for further items, some detailed I/O applications (perhaps collected from users) could be provided and even hardware (e.g. temp probes, light pens, etc.) described or made available. [If you have created any little add-ons for this or any other applications boards, let us at SWN know about it. We will let others know.-ed.]

Boy, This Z is HOT!

by Fred Nachbaur

You may have heard of HOT Z and its decendent, HOT Z-II, and wondered why you haven't seen any reviews on it. I've noticed that other publications are promising reviews "in the next issue," so I gather others are having the same problem we are; HOT Z is a very difficult program to do justice to. It is more than "just a program"; think of it more as an alternate operating system. Think of it as a package that turns your humble ZX81/TS into a full-fledged Z80 development tool. Think of it as being the envy of every other Z80-based machine and you'll start to gain some insight into the scope and usefulness of HOT Z-II.

I don't say all this lightly. While we've been reviewing only those products we feel are worthwhile, there are products for which even the highest praise seems inadequate. HOT Z-II is such a program. How does one review such a thing? What reviewer is up to the challenge? Well, once again Tom (always the sucker for a good challenge) has decided to write the much-deserved review. But meanwhile, I'll try to get some of the superlatives out of the way so he can spend more time on actual description. So consider this "not-a-review" a personal testimonial, or overview, of HOT Z-II; the actual review will appear in the next issue. (Heard that somewhere before, have you?)

So what could make me so excited about a software package? The analogy used by the author to describe the original HOT-Z was something to the effect that it "turns your ZX into a high-performance hot rod." To continue the analogy, HOT Z-II turns it into a blown fuel racer. Your command over the Z80 is complete; on other machines, it would replace a ML monitor, full assembler, debugger/single-stepper, disassembler, and several other highly useful utilities as transfers and code relocation. If writing machine-code has struck you as a colossal hassle, HOT Z-II will change your mind quite completely; entering and de-bugging assembly-level machine code is virtually as easy and as "friendly" as working with BASIC using the good ole ZX ROM.

I demonstrated the program to three assembly-language programmers for other machines, and all were floored by the program's capabilities. One fellow, a long-time TRS-80 Model I programmer, almost couldn't contain himself; "O-migod, an interpretive assembler!" was his first reaction when I demonstrated entering/editing code using Z80 mnemonics. While not quite accurate, the analogy is a good one. Unlike most assemblers where you first enter the source code, then assemble into machine code, HOT Z-II assembles each instruction as you enter it, and then immediately disassembles it! You see the new hex object code appear just like magic! There's also extensive syntax-checking as you enter your ML commands; just like your BASIC, an error cursor appears to mark any boo-boos. Another programmer, who specializes in 68000 development programming, walked off with a dazed look in his eyes after I took him on a brief tour through HOT Z-II. I don't think he'll ever recover from the shock he got when I told him it only costs \$25.

Believe it or not, there is no other product quite like this for any other machine. If you have a Sinclair/Timex, you NEED this program. If you don't, you should buy a Sinclair/Timex simply so you can use this program. If you're even the slightest bit interested in machine code and full control of your CPU, HOT Z-II should be the first thing on your list of "things to get." Only if you're perfectly content to stay in the BASIC ROM will HOT Z remain idle.

Many of us (myself included) started out learning about machine code with the help of the Bug-Byte ZXAS and ZXDB programs. While ZXAS is quite good, and I would still recommend it to beginners, ZXDB is (in my opinion) almost useless; if you've successfully used these programs you will be quite amazed at how much easier it all becomes with HOT Z. There is simply no comparison. The thought that went into HOT Z is pure genius, and HOT Z II is even better. The care and dedication invested in the project is obvious from the moment you unpack that big envelope and find truly massive amounts of documentation. After having the program for several months, I'm still finding new things to do with it; and surprise, it's all there in the documentation. Like Tom Woods' stuff, the manual to HOT Z II is a formidable work in its own right; unlike some documentation which is mostly filler, the HOT Z manual is packed -something you actually have to READ, and re-read, to get the most from this cornucopia of ML programmer's utilities.

I could go on and on. But instead, I'll let Tom do the hard part and simply close off with this recommendation:

BUY HOT Z...BUY HOT Z...BUY HOT Z...

Get it from Sinware, Box 8032, Santa Fe, NM 87504. Cost: \$25 + \$3 S&H. TS2068 users: don't feel left out; HOT Z II is now available for your machine also !!! (same price, too!)

Oh Really? How Hot is It?

by Tom Bent

This comes as a sequel to last issues not-a-review on this program. It became apparent to me that I was not the only person that had to reread the HotZ manual in order to absorb the necessary information. When I got my original version 1.3, about 2 years ago, I was confounded by all the new abilities I had at my disposal. Once I got the hang of using this creation, I rather enjoyed paging through those unbreakable programs (among other things) to see what made them tick. I had all the commands down practically as second nature.

Ray Kingsley (a ZX80er and Sir Clive admirer from way back) decided to one up everybody and redesign HotZ. HotZ II has a completely rewritten structure so as to be relocatable (as was V1.7). No easy feat! A direct memory assembler is also added. HZ II is made for the 2068, but fortunately it was written on and available for a TS1000 too. The minor changes include all commands being shifted rather than one button pushes as Hot Z is. This of course means relearning where all the commands are plus the new ones. Instead of reviewing per se, I feel some hints and warnings would be more appropriate. Indeed, knowing Hot Z is an education in itself.

On loading, the HZ cover comes up. Hitting S will save the 16K (only) version and come back to the screen. Any other key press starts the tour. Location 0000 is first up. Enter 4000 to see your TS operating system variables nicely labeled and use

ENTER to page through memory. A BIG REM is also on the tape. Enter shift Q and load it in Basic. It will come up and tell you it's a BIG REM. You now have a place to write and save your MC routines as well as the whole 16K block (see notes, set RAMTOP, ELINE, STKEND, STKBOT). The 64K version also has an extra name file to load to add meaning to some of those routines. Be careful loading large basic programs, because HZ II does not reset RAMTOP to protect itself. Function 2 in the write mode will (set address to first screen location before F-2). One of the great features of HZ is the "LOAD, look and COPY" (personal use only please) ability for ANY program. I spent a year trying to "break" HZ I in order to boot it on another system (CAI stringy floppy at the time). I couldn't, but Ray has since removed the muss and fuss and I now have HZ II on the Compusa disc drive.

The power of this program will make you a much more prolific machine coder. With the power comes the headaches also. You must stay aware of the mode your in and where END is, as you write (especially insert and delete) your code. Move END around often (it is very useful) and try to keep it as close as what is practical to where you are working. Keep your data and op codes separated. This will help you relocate your program easier if needed. HZ will keep track of relative jumps so that if you insert some code, it will change the relative jump to a regular jump. Be careful changing code back because HZ is a memory edit. You can overwrite an adjacent instruction accidentally (set END properly to avoid this).

Stalking the Lost RAM

by Tom Bent

Paul Hunter of 1630 Forest Hills Dr., Okemos, MI. 48864, with whom I've had the pleasure to converse, is a very congenial chap (English accent and all). He has designed an unique add-on board (written up in various magazines), in particular, the Nonvolatile RAM Board. Based on the 2K 6116-P3 chip (low power CMOS) and a lithium battery, this becomes an excellent system for program development.

One problem haunting both pro and fledgling ZXers alike, is the inevitable crash. With a little forethought and a Hunter Board, this could be a much less terrifying experience. The board comes equipped with a CPU reset switch and jumpers galore to allow flexibility in memory location, power supply and type of memory chip used. The CPU switch resets the CPU to USR 0000 or a 'cold boot'. The Sinclair however, does not use all available ram. On a restart you lose only 16-32K. If you have more than 16K memory, then the extra memory is spared from the crash (not withstanding a runaway LDIR, [or other strange code! - ed.]). If you take the time to make a copy of your program in high memory and with a download routine on the Hunter board, then a reload is as easy as a RAND USR.... A good example of this is HOT Z and its download routine.

Jumpers! You want jumpers! Well here they are. Jumpers on this board allow for either battery power, power from the ZX/TS or an external 5V supply. They also allow the board to occupy any 8K memory block up to 32K. You need not have a full board. You may pop out a chip or two to allow coresidence of an EPROM (on another board) for a special operating system. There are also jumpers to configure the board for NMOS ram or 2716/2732 EPROM as well as CMOS 6116. They may also be mixed! With a little doing, the board can house different types of chips simultaneously. I don't think too many of us want to make the changes involved (apparently Paul doesn't either), and I advise getting the second board (unadvertised special), for \$20.00 with everything necessary for EPROM.

Paul gives several routines in his documentation for you to put in your new found ram. There is still plenty more left open for other things though. One other convenience he has included is a write protect switch (consists of an additional resistor). Once you enter that MC stuff, you can't easily wipe it out. Paul notes that occasionally an unpopulated (with CMOS chips) board may have display problems (an apparent capacitance mismatch). As luck would have it, mine acquired this syndrome as well. The fully populated board was alright. A quick call to Paul brought a replacement chip, 74HC139, an high speed CMOS decode chip (substitute for the LS version). These boards are the absolute highest quality that money can buy. When I first opened that mail pouch I was sure I got my moneys worth. I am really looking forward to Paul's next great achievement.

HZ also has a single step mode. This gem lets you test each instruction separately and check the values of each register and status flags. You can change these values quite easily if necessary. A good way to crash is to start at a rom routine and start running calls (a way to get through those huge timing loops, etc.) You will invariably RETURN with nowhere to go so be careful. Don't POP IX or JUMP IX either. The IX controls the video display. Some operating systems may have these commands in them.

When you SAVE from HZ, set END to the last byte you want to keep and the cursor to the first. Execute a shift Q from the write mode. Write down the difference total. When you want to LOAD this data, set END to the number You wrote down and the cursor to the first byte and shift Q again. Set END to the sum value and start loading (shift Q does HEX arithmetic). You can load your code to any block conveniently. HZ is loaded with this sort of nicety. One thing HZ II does not have that HZ I did, is a command list. (this list has since been added to the 2068 version) Sorry, no room for it in this version. The 64K version does have a name file for some of the HZ routines. Another feature is the alternate name file capability. On the TS1000, the rom name file can be loaded an set so that when you're pageing around the rom, you can see routines by name. Instructions on how to change name files (ALNA) are quite complete as are most of the functions. I suggest writing the necessary code to make the change and use the transfer command to move it in place (just good practice).

Along with the Read, Write and One Step modes, there is a Floating Point mode available. This mode allows you to look at those RST28 codes using the forth like terms of the calculator. Along with these one byte codes come those 5 byte numbers of the successive approximator (for sine, cosine, etc.). This byte appears often and can cause a tremendous slowdown of the screen display and print a lot of gibberish when accessed at the wrong time. Fortunately there are commands available to turn off this beast when not needed and invoke it when necessary (shift 4 and W in read mode).

There are several commands that are meant to be used in series with other commands. For instance, before you load data from a tape, you should clear the area between the cursor and END with Function 0. Relocating code, again needs several steps. There are 9 variable slots (all named) in the operating system calculator stack. These must be set for code, data and variables depending on what you are relocating and execute a Function 8. Function 7 will readdress jump tables. If your routine prints to the screen, then an alternate screen can be set up in memory to watch the action. There is so much to learn in this program that it cannot be presented in just two pages.

The 2068 version has added commands above what the 1000 has. Tom Woods tells me that his 2068 version will load in 1000 tapes. He didn't say if it converted them to ASCII or not though (it doesn't). Of all the programs I have ever used, this one has captured my curiosity and is keeping me intrigued! Still!

Nissim's Marvel

by Fred Nachbaur

I'll start out the software part of the reviews with a package you may have seen the ad for and been a little leery of: a software-only high-resolution program for the ZX81 that sounds too good to be true. After all, how could it possibly do the same thing as those \$100 add-ons that turn your TV screen into a map of, say, 256 by 176 pixels? What do you REALLY get for your \$20; another hyper/meta/super-graphics curiosity, or can you actually do something with it? Well, to make this review a real cliff-hanger, let me answer that with a definite "that depends."

Before I go any further, I should give a brief description of how it works and what options this program allows. The first routine that you call after LOADING makes space in the program and inserts a line 3 REM statement some 5800 bytes long. This becomes the high-resolution display file, and consists of 174 lines of 32 characters, each terminated with an end-of-line marker. The whole program with display file takes about 7.6K, so you have only about 8K for your programming in 16K.

Since all of the routines are defined as BASIC variables in the basic demo supplied, you can just delete the demo part of the program and keep the variables definitions to make it easy to use the program from BASIC. So to get into hi-res mode, RAND USR HRDF and to get back to the normal display file you RAND USR DF. You can have information in both "screens" at the same time, and toggle back and forth at will (and impressively fast) from within a BASIC program. Unlike the normal display file, the HRDF isn't cleared when you enter an immediate command (a nice feature during debugging). Programs will run fine in either high or low res mode, as long as they don't mess with system variable D-FILE, NXLINE, or some other things. You can PLOT or UNPLOT on a 174 by 256 grid, DRAW individual line-characters, ("characters" that are 1 bit high and eight bits wide), PRINT standard ROM characters to the screen, and PRINTC, which allows you to make your own custom 8-by-8 bit characters. Yes, "sprites" (though I use that term loosely) on a ZX81 with no extra hardware. You can also scroll up, down, left, right.

Ah, but wait; the PLOT thickens. You can't just plug this into your application program and mess around for a couple hours and have something that "looks like IBM." For one thing, the hi-res representation is not perfect, being derived from available ROM patterns; not all combinations are possible, and the program gives you the closest pattern it can. So pixels will be in the wrong place or missing sometimes. Also, characters won't quite come out quite like you defined them; you have to get practice and experiment to get a bit pattern that looks good for a given starting line (though you can print the characters vertically at any of the 174 positions, they look a little different at each of the eight starting lines in each "normal" line). Continuous PLOTS sometimes look raggedy, but can be substantially improved simply by running the plotting loop several times (4 repeats usually is the most that will result in visible change). But the final product, "doctored" or not, looks better and conveys more information more pleasingly than

the 44*64 block-graphics of the ZX81 ROM. The exception is the standard ROM characters, which have a decidedly "funny" look (though they're usually quite readable).

The PRINT routine works like this; you POKE the X and Y positions, then you put your PRINT string in a REM statement (without quotes) and follow by RAND USR PRINT. This works fine for the usual legends, etc., but it won't print strings or numeric variables. There is a way to do it (not covered in the docs), but involves messing with NXLN from machine-code; you have to convert any numbers to strings, then POKE (or LD) the string into a pre-determined "buffer" such as a REM statement or BASIC variable before calling your subroutine to set NXLN and call PRINT.

Another drawback is, if you're using the program entirely from BASIC. Though you can draw circles, etc., and can use FAST mode if you have to, it seems to take 4E4 to plot the DEMO display, or any other pure-basic attempts. Don't forget, you're plotting eight times as many points, so it takes eight times as long to fill a given length of screen, 64 times as long to fill a given area using PLOT. ALSO, PRINT and PRINTC are slower than the usual PRINT statements. But if you call the HR routines from machine code, the result is quite impressive indeed. I wrote a few routines to draw axes, markers, and the like, and draw in a graph of some externally obtained data (voltage vs. time), and the whole show happens in a couple seconds, including data scaling, over-range checking, and plotting routines. Plotting a line across the screen is actually faster than calling the standard ROM PLOT/UNPLOT routines! So, if you're excitedly getting into assembly-level machine code, this is an ideal package for you to work with; you'll find tearing into it as challenging as any adventure game. But if you're mainly interested in dressing up your BASIC programs, you might be a little disappointed by the slowness of the result.

But assuming you can assemble your own graphics routines (see Harry Doakes' series in SYNC for some neat routines you can adapt to this, and expand from) this package, after some work of your own, can impress the hell out of people who will be looking around for some mysterious black box, since "everyone knows you can't do hi-res with the Sinclair's limited hardware." The fact that it was done is quite an impressive feat, and it is no surprise that the programming is somewhat unorthodox. In a sense, it's a "dancing dog" program, but this particular pup can dance pretty darn good. It does some funny things with the stack, messes with system variables like D-FILE, NXLN, and others, and jumps to "hidden" entry points. Don't EVEN attempt to make sense of the program without HOT Z, preferably HOT Z II, so you can make changes easily; some of the code could be optimized, the package to date appears to have some rough edges which should settle out as time goes on. Example: the CLS routine scrolls the display up; very slowly. With some experimentation and thought it isn't too hard to write a fast and "neat-looking" CLS by filling in the 21-3/4 lines of 8 smaller lines with the right values for a clear screen. (No, they're not all zero and not all eight lines the same.)

I have spent a great deal of time with this program and HOT Z II in an attempt to re-locate the SHR program and display file into the 8-16K area. Even after getting past the "traps" and other

oddities, and even with the help of HOT Z's potent "RELOCATE" command, I haven't had any success yet. Though I can move the display file around in the 16-32K range, something hangs up when it goes "downstairs." Hopefully someone will break this barrier, as it would free up enough space for BASIC and machine-code applications programs of reasonable complexity. It would also allow EPROM versions, say in a Hunter board with the EPROM in the first slot and CMOS RAMs (for the display file, variables and program data) in the other three. What do you say, Nissim?! How about re-writing it into a Hunter-board-able version?

Is this program something for you? If you're interested in the highest possible accuracy, you may not be too happy; you might get more out of Russell's HRP package (reviewed last issue) which gives you an exact bit pattern (albeit only on the printer) and includes point-to-point plotting which is very often useful in scientific work. Although Nissim's SHR has a COPY to printer option, you get the same distortions on the printout as you have on the screen. If you really need "the mostest" you should maybe pop for a hardware add-on such as Memotech's HRG or (better yet) JLO's TI video upgrade board. But if you'll settle for "semi-hi"-res, this one will do it for you. Some pretty

radical games could be written around this program, perhaps with the help of a compiler. For math, technical and other serious work, it can serve quite well to increase the information density of your graphs when coupled with machine-code "drivers" for speed. What do I think? I recommend it highly. Even though it's hard to disassemble, being riddled with protection schemes, it has proved a fascinating subject for study and experimentation. It is, after all, a "first," and a certain raggedness is to be expected of all major new releases. It is certain to undergo further evolution as the author and users discover refinements. Order from Nissim Elmaleh, 5100 Highbridge St. 53, Fayetteville, NY 13066 Cost: \$20 + \$1 S&H.

By the way, Nissim Elmaleh also has a high-res machine-code driven word-processor program (\$25), that actually gives you lower-case letters on the screen! Again, they're a little off-beat looking, but they're definitely lower-case. Interestingly, the distortions don't show up in the printed output (TS2040 only at this point) which looks as good as either of the other major software-only word-processors on the market. You can print various ways, and enough options are provided to make it a usable tool for TS2040 purposes.

Byte Back at the Real World

Finally! A product that is easy and straightforward to review. I'm speaking of the BB-1 control module from Byte-Back, Rt. 3 Box 147 Brodie Road, Leesville, SC 29070. It is easy to review because it is easy to understand, easy to build and easy to implement. When you open the package, you find all your parts neatly organized into envelopes, with a comprehensive set of instructions that will guide you through the steps to turn that collection of parts into a working I/O board. When you're done, an evening later, you'll have a nifty little board that gives you the heart of any number of dedicated control applications. Basically, the device consists of two PIO chips, (one for input, one for output), eight reed relays for output control, and the necessary decoding and buffering circuitry. Because of its two I/O chips, the BB-1 is almost like two boards; not only can you control up to eight different circuits or devices, you can also get the logic status of eight input circuits or devices. A nice touch is the on-board regulator and heatsink. Other nice touches are the plated and masked board, on-board status LED's for output indication (just great for debugging), and a competent yet uncondescending attitude in the manual.

The board is decoded at addresses 16381 (for input) and 16382 (for output). PEEKing 16381 gives you a number that, on conversion to binary, gives you which input lines are pulled to ground (active low). Similarly, POKEing 16382 with the desired number actuates the relays and indicator LEDs. Byte-back warns against having anything else in this memory range, but I've tried it on two 64K RAMs and on the Hunter board, and it works flawlessly even if that block is already in use by RAM. Of course, you can't put anything else in these addresses, but I've found no problems due to the apparent conflict. Additional boards can be decoded for other addresses, to give you as many lines as you'd care

to wire up. The output circuit has an on-board latch to keep the output status the same even if you NEW the computer; only a POKE or LD into 16382 (3FFEh) will change the status. The on-board relays are the tiny DIP reed variety which are fine for controlling low-level currents like other TTL stuff, analog summing amps, etc. but should be used to control other (external) relays to handle loads over about .5A resistive, and any non-resistive loads. Or use power transistors or triacs (for DC or AC respectively). You could replace the relays with opto-couplers if you need the faster response.

This board (\$59 kit, \$69 A&T) is a good buy for those who wish to get into outside control and aren't afraid of the basic electricity aspect of it, but would rather not get into decoding, PIO, latches, buffers, drivers and all that sort of thing; an evening of pleasant kit-building, and you're ready to hook it up to your bells and whistles. We do recommend that you get sockets for the PIO (and other) chips just in case you blow one out; Byte-Back doesn't supply them. I got lazy and soldered mine in, and haven't had any trouble (Yet). You'll also need an edge connector to the relay and input lines, unless you plan to hard-wire the board. Other than that, there are no surprises. You wire it up, it works. Now what you do with it is up to you; if you can turn it on and off with a switch, you can turn it on and off using your computer.

EDITOR'S NOTE: So what am I doing with it? As I mentioned in the intro, I'm building a machine to measure battery performance, based around a TS1500. The BB-1 is used to select the charge/discharge current (first 5 bits), charge or discharge mode (6th bit), turn the cassette motor on and off and select VOTEM mode (7th bit), and select LOAD or SAVE (8th bit) to prevent feedback or ground-loop problems with the recorder.

8K ROM Upgrade

By now you should have a pretty good idea of what I've been doing; so what's ol' Tom been up to? Well, instead of a much-deserved rest from high math writing, Tom has been busy upgrading his hardware with the Oliger system. Starting out as a self-described "electronics duffer," he has succeeded in getting through all the pitfalls and has an enviable system built up. Included is the facility to burn 2K or 8K EPROMs. The first task to which he's putting all this neat hardware is to upgrade the ZX 8K ROM. He's burned a couple prototypes and has figured out an easy way to incorporate the new EPROM inside the machine in place of the old ROM.

So far, the changes we've come up with are:

- 1) Correcting the LPRINT bug and the division bug.
- 2) Changing the bit patterns of V, W and Q in the character set to make U, V and W easier to distinguish and end confusion between O, 0, and Q.
- 3) Changing the initialization routine so the machine comes up in FAST mode after NEW or reset. Also, the machine automatically goes to FAST mode when breaking a BASIC program and hitting ENTER to get the automatic listing. This bypasses the sometimes annoying "reformatting game" that the existing ROM loves to play.

We're also determined to improve the SCROLL and possibly CLS commands (these have been improved-ed). SCROLL in particular could use some work.

By the time I get out this issue, Tom hopes to have his little EPROM production line in full swing. Contact him about getting an EPROM of the corrected ROM, and instructions for installation. If you've seen the reports of various bugs and wish you could do something about it, well now you can.

Olio a la Oliger

Reviewed by Tom Bent

SWN being a technically oriented magazine, it is very fitting to review various hardware efforts reported in other magazines, that may or may not have caught your eye. The unfinished series started in Syntax Quarterly Magazine Winter 1982, by John Oliger, 11601 Whidbey Drive, Cumberland, In. 46229, stands out from the crowd as far as hardware projects go. John presented articles describing how to build several different boards; parallel printer interface, expansion board, power supply, 8-16K EPROM board, EPROM programmer and last and absolutely the most, a 64K memory board. This isn't the end though. Other available works include a 2764 EPROM programmer and a read board, inverse video, power supply, and a memory monitor readout (led). He is presently shipping his TI video upgrade color board which will bypass some of those Sinclair crash and display quirks. [Review next issue.] This is a nice add-on with HI-RES and 40 col screen possibilities.

Master Your TS

John Eddington

I can recommend the Bantam book (paperback, March 1983 @ \$3.95) "Mastering Your Timex/Sinclair 1000 Personal Computer" by Time Hartnell and Dilwin Jones. If a convention were made that "A" is "Attracted", "B" is "Beginner", "C" is "Coming on well", "D" is "Dependable" and "E" is "Elegant", then this book is for A through B to most C. All the basic functions are covered clearly with good illustrative programs which work, and chapters on "Business Uses", "Converting other BASICs", and "Moving Graphics" are easy to refer to. Throughout there are useful tips on "How" and on style. At that price, what is there to lose?

Lprint With RADIO SHACK Paper

John Eddington

Radio Shack's Thermal paper for their TP-10 (#26-1332) at \$5.95 here (Canada) fits the Timex 2040 fine with a bit of squashed corrugated paper at each end. CAI's paper for their 40 char. printer is the right size. Both are considerably cheaper than the Timex paper, and easier to obtain.

OFF THE WALL... The Memotext files for this entire issue just barely fit on the two sides of a single C-60 cassette. (That is just issue 1/4. This entire book is over 200 minutes of LOAD time.) Just another answer to the question, "How long is 60 minutes?" How many individual operations does the system have to execute to get this all to paper? You figure it out. I can't count that high...

Instructions given in the three SQ issues are vital to being able to construct most of the boards, so if you don't have them, get them. Details include how to make your own boards, but considering JLO's price, you can't go wrong by buying his. One drawback to these boards is the number of feedthroughs that you must solder in. There are no plated holes (some are now plated through). On the five boards that I built, there were two undrilled holes that had to be fixed. The 64K ram board I received was slightly warped, but this turned out to be a plus. It fits so snugly in the connector that there is absolutely no wobble! Aside from details on how to make the boards, he gives full schematics and theory of operation as well. For those of you that tried his little modification to enable execution of MC in the 32-48K block and found it didn't work (SQ p. 47 summer 1983), well try again. This time hook up pin 1 and 13 of the 74LS10 to +5V. There apparently was a typo in that issue. (Also, it won't work with all 64K packs.) Not only is this a hardware project, but it is a software project as well. The printer driver and ASCII conversions must be entered in MC. He suggests HOT Z for data entry, and gives some how to's in his documentation, for first timers. I recommend HOT Z as well. HOT Z-II is now available to make full screen editing possible (more on this next time).

The 64K board is really something else. It gives a full 64K, by bank switching the first 8K block with the 8-16K block, decoding either into the 8-16K block. [The actual 0-8K block is unused in most "64K" Rams; in JLO's it is an alternate 8-16K block. -ed.] If you leave your machine on, it is possible to load those operating system eeproms (if the eeproms are removable from their boards) into ram, and switch between two different systems occupying the same space! You may also switch out this area in 4K blocks and switch out the 48-64K block. This block is for holding basic programs on 2764 for fast download (I didn't build this one yet). This board also adds an extra control line, Port Sel not, to decode the printer port (FFFF). In constructing this board, I came up against a seemingly insurmountable number of feedthroughs, but survived none the less. JLO says "90 minutes for this task." However, I would suggest you plan on making the evening of it. If you goof, it could take a while to trace. The assembly instructions were better for this board than any of the others (thank god). For this reason I suggest tackling this one first. One other great thing about this board is the price. \$19.95 bare board, \$34.95 kit, not including memory chips. He gives locations for getting the pin 1 refresh 4164 chips. Mine came Federal Express the next day from Microprocessors Unlimited for \$53.00. Total cost \$88.00 plus a good learning experience (the chip cost is now 1/2 of this price, and you could build this 64K board for about \$60.00).

Most of the boards require the expansion board (\$10.00) which has the extra port sel not line on it. This board will house 4 boards (the big expansion board is no longer available). This compact board has room for 3-20 pin sockets as well.

You will need at least 6 edge connectors for this series unless you plan on direct solder connections. I found Sintec, in New Jersey, to have the best price at \$2.44 each in lots of 10. Remember, you need the extra pin out for port sel not.

The 8-16K eeprom read board was the easiest one to construct. JLO suggests two of these. One for 8-16K and one for 0-6K. This second board is for system mods. You can now start up with a full 64K, repair the LPRINT bug and have your printer use the COPY, LLIST and LPRINT keys. The programmer board was the least explained by the "how to" so be careful building this one. This board has no verify circuitry, which is fine with me. I wouldn't want to make a living programming eeproms with it though. This circuit has it's own timing circuit which must be fairly precise. It calls for a precision cap and resistor. I used a 500K trimpot and set it with a DVM to 453K. I couldn't locate a precision 453k resistor anywhere. One other change from the schematics is changing a 180pf cap to 330pf to help with the timing a little.

I don't have my printer port working yet (still waiting on one chip) so I won't comment about it at this time. JLO also offers parts kits for all of his boards. I suggest getting them to save you time and money, as his parts prices are as good as his board prices. This package is an entire system. It is well thought out and priced in the ZX users pocket book range. It is nice to see this type of expertise available in this price range for Timex Sinclair users. We will update this system from time to time as the boards go into service. Watch for major articles by John Olinger. They will be forthcoming, somewhere. Thanks John.

Delphic 2K Toolkit, Tape Version

Had it been possible for SWN to adhere to its originally intended publication schedule, you would have seen a review on this toolkit in the Nov/Dec issue, and a new product announcement and ad in the next issue; this one would have contained the update announcement. But things didn't work out that way, so now you get the whole story in one installment.

The story started when we heard of a toolkit EPROM (2K) from Delphic Enterprises in Corpus Christi, TX. Sure we'll review it, I said, but since at that time I didn't have a Hunter board to plug it into I had Pete Hoffman (the author) send it to Tom, who made a tape copy for me. I liked it from the moment I called it the first time; I was much impressed by the program, and surprised that a tape version wasn't available for people who don't have a Hunter Board, or don't want to "mess up" their pristine CMOS board with an EPROM, and haven't broken down for a second (EPROM reader) board yet. Well, surprise- Delphic was not interested in getting into the "tape wars," and generously offered tape distribution rights to ... you guessed it; SyncWare News. So, all you RAM-fans, you too can have this well-designed tool at your disposal, tucked away in any 2K niche you have available (in 16-64K RAM).

The toolkit is menu driven from a single USR call. The program is fully relocatable, and will run anywhere that you can run machine code. When you

call your selected starting address and get the menu, a single key-press (logically coded) accesses any of the eight routines. The first of these rennumbers BASIC programs. Yeah, sure, I hear that collective "ho-hum, another renumberer." Renumberers have been around since day one, some ok, some not so OK. But this is like no other re-numberer I've used; it will renumber only a PART of a program, or the entire program, unlike other TK's that make you renumber everything. Finally someone did it right! Partial re-numbering carries with it special problems; such as GOTO/SUB references, and user error. But every eventuality I can imagine has been foreseen; for example, references are changed (or not) logically, and "goofy" references are flagged with 9999. Even if your re-number increment makes the section run into the next, the program will save the situation by re-numbering in increments of 1 until the numbers are "caught up." Additional renumbering fixes up any oddities. Of course it changes GOTO/SUB references; a "re-numberer" which doesn't is really only half a re-numberer. Related options are to DELETE an entire block of lines, COPY them to another part of the program, and (by using delete after copy) MOVE blocks of program lines elsewhere in the program. You can even MERGE two BASIC programs into one by using PRESERVE and COMBINE, and there are no space limitations on use other than available memory space.

What else? Well, one I've never seen in other TKs is REM KILL; as the name implies, it deletes all the REM statements in a selected block of lines (or the whole program). Handy for when you're almost done with a program and have to make room for those last few "polish-up" routines. Another option is UNLOCK, which aborts auto-run programs after loading so you can back them up. A basic necessity is a FREE-SPACE routine, so that's in there too.

Other features: All inputs are prompted and verified on the screen. Control is returned in FAST mode; when programming you'll rarely have the patience for SLOW mode. Lastly, it is very well error-trapped and disaster is unlikely even with user error. If you realize you just hit the wrong function, you can bail out before DELETing something. That would be enough for most folks, but Pete Hoffman is one of those people who won't stop until he's pynched every last byte. His 2K toolkit includes yet another highly useful option: a string search routine. You can search for any character (including functions, keywords, and other tokens) or string of such characters. You could, say, search for LET G\$= to see every time you assigned to G\$. You get a report of every line in which the selected character or string appears. (Including lines containing machine code, but of course you can't search for those codes that don't have a corresponding display character). Great for economizing memory and general debugging. I mentioned an "update" at the outset of the article; one of the updates has been to economize memory even more to allow this routine to print the entire line in which the string appears, along with the line number. Another improvement in the update is at the end of the routine; depending on what you do during a 3-second pause at the end, you can get back to the BASIC listing, back to the toolkit menu, or let the PAUSE expire to copy the screen of toolkit parameters to the TS2040 printer.

A tape version has special requirements, but also offers additional flexibility not possible with an EPROM. So it was necessary to write a driver for the program to make it easy to locate and use. An

additional routine is provided to transfer the program wherever you select. The tape version also contains a check-sum routine independent of the toolkit which checks for proper loading initially and which you can use afterwards from within your program. The program won't let you put it somewhere silly, like anywhere it would run into a boundary. Even if you try to put it in the 32-48K range, you are asked if your machine has be "Oligerized" to verify that you know what you're doing. Also, the instructions were expanded expressly for the tape version user.

Here's how can you get this excellent product. (Famous "bottom line"...). If you want the EPROM version (for semi-permanent installation in your Hunter board or ROMPAK) you should send your check for \$18.50 (\$20.00 foreign) to Delphic Enterprises, PO Box 72205, Corpus Christi, TX 78472. But if you want the tape version, don't order from Delphic; instead, send a check for only \$12.95 (\$15.00 outside North America) to Thomas B. Woods, PO Box 64, Jefferson, NH 03585.

[EDITOR'S NOTE: Since this appeared in SWN 1:4, Mr. Hoffman has completed the 4K expansion of this program. In addition to the features in the 2K version, he has included several other options that make this by far the most comprehensive toolkit available anywhere. New features include: REM generator (you select which fill character), NVM storage (to boot part or all of a BASIC program into the unused part of your 8-16K range), and a one-step MOVE command. Add to this a hex-dec converter and a tape indexer to read the program names on a tape. But that's still not all! My favorite is the VARIABLES READER; as you probably know, I'm one of those programmers who loves to manually define variables to save space; the result during program development is often "variables soup." The variables reader lists the names, type, and contents of the variable area just like the ROM lists the program area. It is simply beautiful. Delphic is offering a special rebate for 2K EPROM customers. -fn]



SYNWARE CO.

THE CASSETTE CONNECTION

With this issue, we start a "mini-series" on improving cassette reliability. I'm sure you'll agree that unreliable LOADs are perhaps one of the most frustrating aspects of personal computing. With a little knowledge about the SAVE/LOAD process, we can take steps to save ourselves a lot of grief.

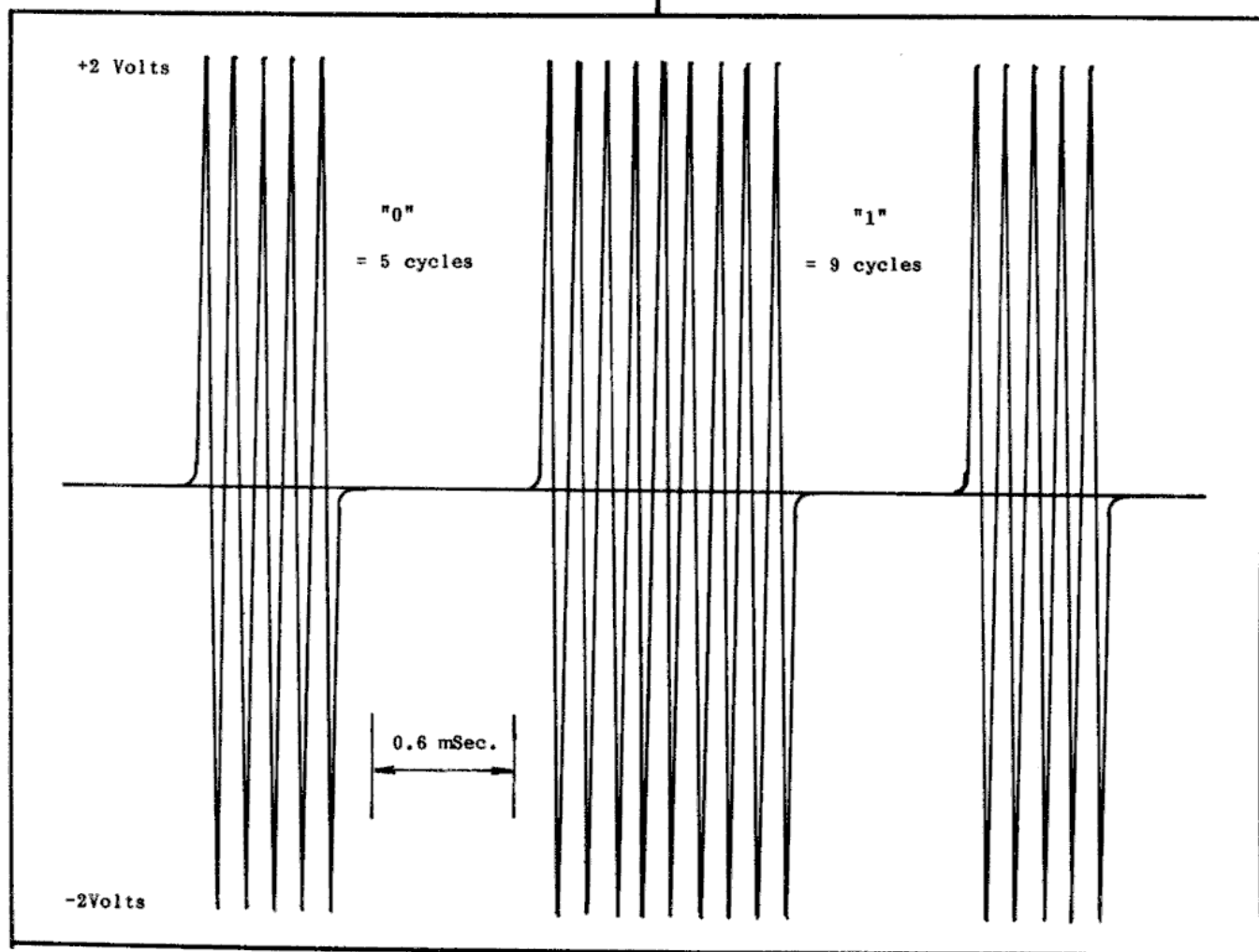
We'll start the series with a discussion of the nature of the cassette signal, and derive various hints (and perhaps get rid of some misinformation) that will help. Next issue we'll pursue a quest for the "ultimate load aid," and will cover various ways of monitoring & conditioning the cassette signal.

TIPS AND MYTHS

First, let's have a discussion of what the LOAD signal actually is. By its nature, the cassette interface is a SERIAL device - each bit is sent one at a time. The simplest way of transmitting serial data is to define a pulse as binary "1," and the absence of a pulse as binary "0." This is not very reliable, and timing is critical. The slightest drop-out will cause false data.

Now, a statement that may surprise you; the ZX/TS SAVE/LOAD approach is one of the most reliable among the "bottom-end" computers. If you think you've got trouble with your ZX loading, you haven't tried loading CoCo or other computers under less than ideal conditions. One reason for this is that ZX loading is relatively slow. Data is transmitted at an average of about 300 bits per second (or a little more than 2 Kbytes per minute), compared to say 600 baud (bits/sec) or more for most other machines. As a result the ZX system is far more immune to bad tape, crummy recorders, etc. Another reason is that exact timing is not crucial for the ZX; our guess, based on actual experience, is that speed fluctuations up to 30% are ignored.

How does the ZX do this? Take a look at the diagram. Instead of "pulse=1, no pulse=0," our machine sends each bit as a SERIES of pulses. The rate of these pulses is about 3300 Hertz. Five such pulses represent a "zero," nine pulses is a "one." Each bit is separated from the next by a silence equal to the length of a "zero." This implies that actual LOAD/SAVE time varies with the ratio of zeroes to ones! I.e. an empty array (all zeroes) saves faster than the same array filled with data.



```

10 DIM D$(10000)
20 FOR D#=1 TO 10000
30 LEFT$(D)=CHR$(D)
40 NEXT D#
50 PRINT "X"

```

This approach is highly immune to differences in recorder /player speed. Even speed fluctuations are virtually ignored, shooting down "myth No. 1" - "your recorder's speed must be dead-on, with no flutter or wow."

Load Hints

1) "Set LOAD level so you have 4, 6, or N bars on the screen." I don't know about you, but I find counting bars a little like counting the stripes on a frightened zebra. Even if you do your count during an "all-zero" part of the program, how many bars you see depends more on your TV than on the recorder. A slightly better criterion is the relative WIDTH of the bars; if the white portions are "fat," level is too low; if the white bars are considerably narrower than the black portions, level is probably too high.

3) "Adjust azimuth alignment frequently." Balderdash. Once it's set, it's set, and it's VERY rare that you'll have to mess with this. You're likely to have more trouble fooling with this, unless you have a KNOWN GOOD azimuth test tape. I have yet to see a machine that needs to have this factory adjustment tampered with. [See also "More Tips and Myths" later on for more info on this - ed.]

```

00010 PROLOG
00020 PRINT=0 TO 0.01 STEP 0
00030 IF PRINT=0 THEN GOTO 0.01 STEP 0
00040 NEXT R
00050 PRFUSE 60 (graphic on "5" or "8")
00060 PRFUSE 60
00070 RUN

```

Mention should be made of how the various available speed-load routines fit in. (See Z XLR-8 review later this issue). These can be a marvelous refinement for your system, but they require a good recorder and very good tape for reliability. If you're having trouble with the regular tape routines it will only be worse at higher speed.

Choosing a Recorder

Look for these points:

- 1) Does it operate from a 6 Volt or greater supply?
- 2) Does it have a MIC jack? (Sounds silly, but at least a few models don't)
- 3) Does it have a tape counter? VERY handy.
- 4) AC adaptor included?
- 5) Does it allow cue/review (F.Fwd & Rewind permitted in PLAY mode for finding selections); quite useful but will wear head faster if used a lot.

I've been using Radio Shack Minisette IX's and have been happy with them. (At that price I'd better be!) The first one still works fine even though the head is about shot after 18 months of daily use.

I'll close this installment with a few words about tape. The tape itself is rarely the cause of bad loads, unless it has bends or kinks in it. More often, the cassette shell is at fault, causing binding which results in the tape being pulled across the head unevenly; the poor contact gives drop-outs (and bomb-outs). I used to use Maxell tapes, but recently I've found that their shells aren't up to snuff. Memorex, forget it. Good tape, rotten shells. TDK, Agfa, Sony, 3M tapes seem to be fine. Bargain "3 ferabuck" tapes are ok sometimes, but don't trust them. Some tapes record "hotter" than others; mark your ideal volume setting with a spot of paint on the thumbwheel, vary it from this setting to accomodate different tapes. Commercial software is a generation removed from the master, and can present special problems. Reputable manufacturers provide at least two copies of each program, under the premise that at least one of them will be LOADable. But ALWAYS make a back-up copy the first time you get a successful load, then put the original away. You'll probably regret it if you don't.

More Tips

Before we get on with our discussion on loading aids, here are some additional comments on recorders and the tape medium in general.

Ray Kingsley, proprietor of SINWARE and the author of the already famous HOT Z and Z EXTRA programs, sent in a letter sharing some of his experiences with the vagaries of the tape medium.

"... I was very interested in your cassette article and generally in agreement. I once had a batch of tapes with lumps on the first few feet near the end of side 2; they caused the biggest rash of returns I have suffered so far. The lumps were a result of an imperfect join on the ring and the lump stretched several layers of the tape wound around it. The problem was made worse by my habit of using tapes that just fit in length, something I've begun to give up. Cassette cases are a real hazard; many batches of tapes have one in 10 that won't load because of the case. However, I do disagree a bit on head alignment. Many of my tested tapes, cassettes that I can load on at least three players, come back and I test them and they load

again, but the buyer can't do it. I think that's usually alignment. (I managed to realign one of my Radio Shack CCR-81's that perversely went out of alignment, it wouldn't read tapes made by its twin sister, with just a not-very-good ear and a data tape, listening for maximum output. Not a recommended technique, but it might beat waiting two weeks for a replacement.)"

Head Alignment

After receiving Ray's letter, I did some more asking around, as it appeared that computer users fall into two camps: those with chronic head azimuth problems, and those who never touch the adjustment and don't have trouble. From the limited sample of people I know with computers, there seems to be less trouble with the miniature "pocket notebook" type recorder than with the "table-model" size machines. This may be because of the physically smaller linkages in the little machines, resulting in less play as the head mechanism wears. Trouble tends to develop as the machines get older and the head and head carriage wears; so take Ray's suggestion (from HOT-Z documentation) and "please do not make things difficult for all of us by using the oldest machine in the house and sticking with it."

If your system is working fine most of the time, don't go adjusting the head azimuth for the heck of it. My reason for recommending against head alignment in the first installment was based on these considerations: 1) if you don't have a known good test tape, it is difficult if not impossible to determine exactly where "dead-center" is. 2) Frequently adjusting the azimuth can introduce added play, as the assembly is rarely very stable to start with. Especially true of machines that use lock-paint to help keep the assembly "tight." The more you adjust it, the more it will need adjustment.

If you suspect head alignment trouble, you can adjust it by ear if you have to; using a data tape, adjust the little setscrew until the sound is the shrillest you can get it; most affected are the high frequencies, so listen for the maximum "edge" in the sound rather than maximum volume, which can be hard to judge using a computer data tape.

If you received a new tape that you just can't load, the problem may be head azimuth on the recording end; in this case, you can do as Ray suggests and adjust your machine to that tape if necessary. But don't forget to set it back before making your backup and other saves, or you'll perpetuate the error and really mess yourself up.

Lumps & Bumps & Backups

Finally, in case you're not already doing it, always avoid the first 30 seconds or so of any tape; almost all of them have some degree of bumpiness in this area, due to the little pin in the hub which holds the end of the tape. This is also the area most likely to get damaged, so consider the first 30 seconds "just more leader" and your problems will decrease. In a real emergency, you can sometimes get a balky tape to load by increasing the head pad pressure by bending the spring outward slightly with a non-magnetic tool. This on occasion will improve head contact enough so you make it over the "whoop-de-do's". If it now loads successfully, make a backup and throw the modified one away, as bad tapes only get worse, never better. Lastly, in case you're brand new to this,

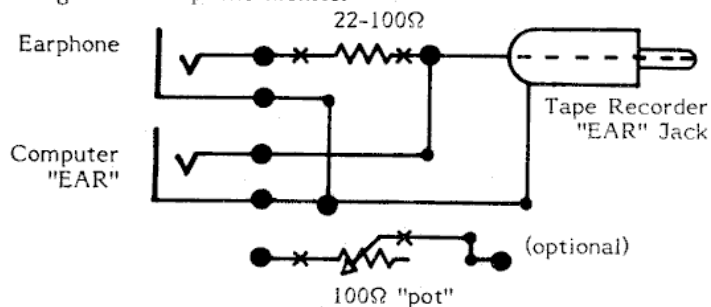
MAKE BACK-UPS, MAKE BACK-UPS, MAKE BACK-UPS!

The Ultimate Load Aid

Even with some knowledge of how the tape process works, you still might have to resort to external loading "aids" to get the 99%+ reliability you need. A variety of products are commercially available to do this, ranging in price from about \$15 to over \$40. At least one fast-load system (Powell Hargrave's SDS) comes with its own loading "filter" as part of the package price. What are these, and what do they do? Is there an "ideal" load aid that will solve all your loading problems? In this discussion I'll try to cover the various approaches that have been successfully used, starting with the simplest and working up to the more sophisticated.

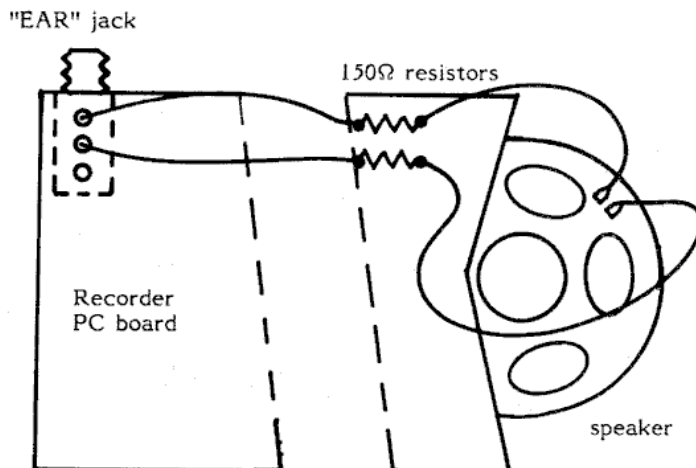
If your hearing is relatively unimpaired, then you already have a highly effective audio-signal analysis tool. This is what I've found to be the "ultimate load aid" -- with a little practice anyone can learn to use their ears to judge the setting of the tape recorder and the overall integrity of the tape signal. But first you have to be able to hear the signal. Sure, you can unplug the ear jack and play the tape; but as supplied, you can't listen to the tape while it is loading. You could buy or make a "Y" connector to the ear jack, with one end of the "Y" going to the computer as usual, and the other end going to one of those cheapo earphones that usually come with cassette recorders. This will allow you to hear the signal while it is loading into the computer. But since the required voltage for successful loads is quite high, chances are you won't be able to live with that nasty screech even coming from an earphone. Solution: place a resistor in series with the earphone to attenuate the signal without affecting the signal to the computer. Typically, a value in the range of 22 to 75 ohms will cut down the sound to where it's still audible but won't turn the entire household against you. Or you can install a 100 ohm rheostat (potentiometer with only two connections used) to adjust the relative volume of the earphone.

Diagram 1. Earphone Monitor



So now what? You've hooked up your "computer to ear interface", now how do you use it? Well, first off you can use it to set level. The best way to do this is to play a short section of the tape to be loaded. Listen to the sound from the earphone as you turn up the volume on the recorder. You will find a point, usually between 1/2 and 3/4 of full volume, at which point the character of the sound starts to change, becoming raspier and even more offensive than the load signal already is at low levels. This is the point at which the amplifier is starting to bottom out. The best volume setting is usually right after this bottoming starts to occur, but before it has gotten so bad that the machine "splatters." Once you've set the volume, rewind the tape and LOAD it into your computer. While it's loading, you'll be able to detect changes in the sound resulting from drop-outs or other irregularities. An earphone monitor of this sort will also help you in finding desired selections on a long tape, and is helpful in trimming up azimuth alignment if you have to. It won't take you long to differentiate between tapes that sound crisp and steady, and those that sound muddy, or fluctuate in level, or (most common) have "highs" that weave in and out (the "crispy" sound getting louder and quieter). Granted, simply monitoring your signal won't make a bad tape loadable, but at least you'll have a better idea of what's going on. Since on most recorders the amplifier output signal is available at the EAR jack during recording (for monitoring purposes), you'll be able to hear when you SAVE as well as LOAD, helping to insure that the tape recorder is indeed connected to the computer. The unfortunate side effect is that many systems will have trouble if the EAR jack is connected during a SAVE; a positive feedback loop is established and the system "howls."

Diagram 2. Internal Monitor



It is good practice to disconnect the cable to the EAR jack during a SAVE.

Instead of using an external earphone, you can install two resistors in your deck to use the internal speaker as an audible monitor. Connect flying leads to two 150 ohm resistors and cover with tape or shrink-tubing. Connect one end of each to the speaker lugs. Place an unconnected plug (open-circuit, not a shorting plug) in the EAR jack. Now with a tape in the machine and in the PLAY mode, touch the two free ends of the resistor wires to the three lugs on the EAR jack; there are exactly six permutations of two wires and three lugs; try each one. In at least one position you will hear a quieted-down version of the signal. Now, try to record. SAVE a program and put the player in RECORD mode. Once again, touch the resistor wires to the ear jack and try the various possibilities. On nine out of ten machines there will be a connection which leaks a little signal to the speaker in both PLAY and RECORD modes; connection to the EAR jack will usually be the center lug and the lug closest to the edge of the board. Since there are many variants in exactly how the switching is done, the exact connection will vary with different machines. Often (as on the Minisette IX) you can simply bridge the two resistors across the board from EAR jack to where the speaker lines go on the board (blue wires on Minisette IX).

In case you're worried about loading - the 150 ohm resistors are so much greater than the 8 ohm speaker impedance that the "leakage" load is negligible. Since the speaker is partially engaged during RECORD mode, some machines might give acoustic feedback if you record with the built-in mike. If this is important, increase the value of the resistors until the howling stops. Diagrams 1 and 2 below summarize these acoustic monitors.

The Optic Approach

There are folks who just can't bear to listen to those computer screeches, attenuated or not. For you, and for those with impaired hearing, an optical approach may be better. Circuit 1 shows a way to do this with LEDs. This circuit also gives some clipping action and helps flatten out the tops of the load pulses. The 10-ohm resistor is not really vital, but some machines have enough power to blow

out the LED at high volume. this resistor helps limit maximum current and extend the lamp's life. (It also tends to enhance the clipping action somewhat.) For further LED protection, shunt 3 1N4001 type diodes in series across the LED. Unlike the acoustic monitor, this circuit usually won't show the monitor signal during recording. This is because the level at the ear jack is usually lower in record mode, and cannot be changed with the volume control. It therefore never reaches the 2V required to light the LED.

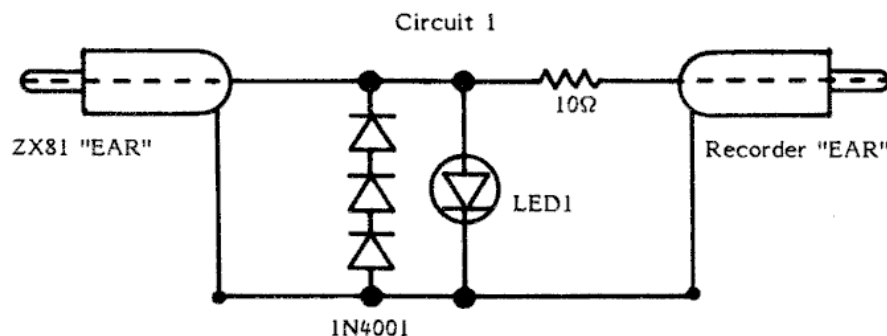
Circuit 2 shows a recommended switch addition which will save you from having to pull the EAR plug before a SAVE. Points of insertion for circuit 1 and circuit 3 (discussed next) are indicated. Switch pole "A" in circuit 2 may be omitted on most machines (allows DPDT switch).

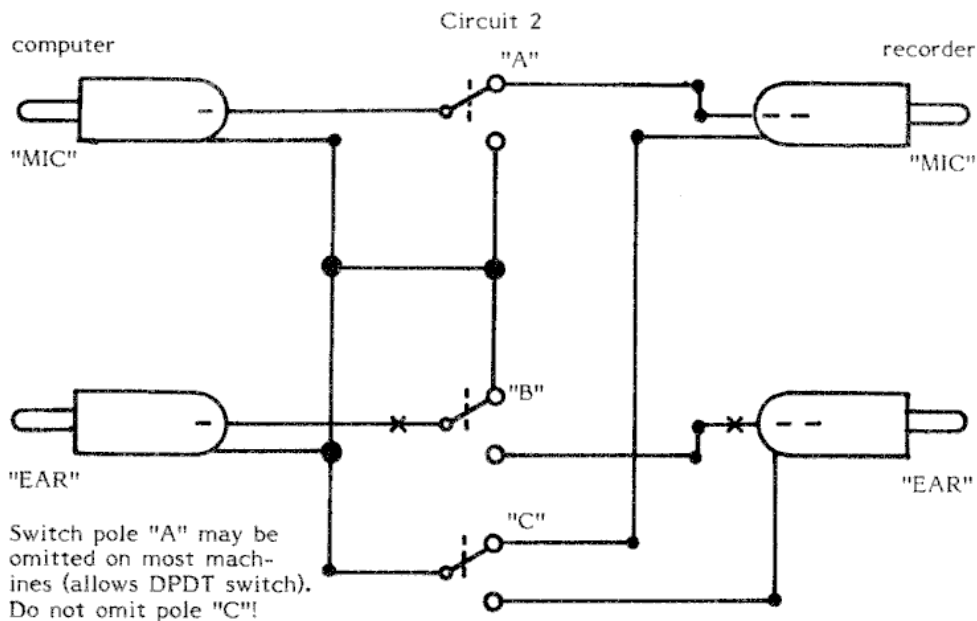
A Tape Conditioner

Now, we can at least monitor what's going on. How about conditioning the signal somehow to improve reliability? Various approaches are possible. As mentioned above, even the simple LED circuit helps out a little by clipping off anything over about 2V peak (or 4V peak-to-peak). The best solution is to use any one of a class of circuits called "comparators," which basically "square up" an analog signal to make it more acceptable to digital equipment.

Comparators are essentially high-gain amplifiers that are designed to be over-driven, and usually have some positive feedback from output to input which latches the output in either of the two "overdrive" conditions, full ON or full OFF. The output stays OFF until the input climbs above a certain "upper trip point," and then snap ON and stay there until the input drops below the "lower trip point," at which time it snaps OFF. Result - nice, constant level square waves no matter how the input signal meanders up and down.

The span from upper to lower trip points is called the amount of hysteresis of the circuit. With no positive feedback, the upper and lower trip points are the same, and the output state changes whenever the input crosses the reference line. The disadvantage with this is that it can result in extra pulses if there is a glitch riding on the leading or trailing edge of the input signal. The

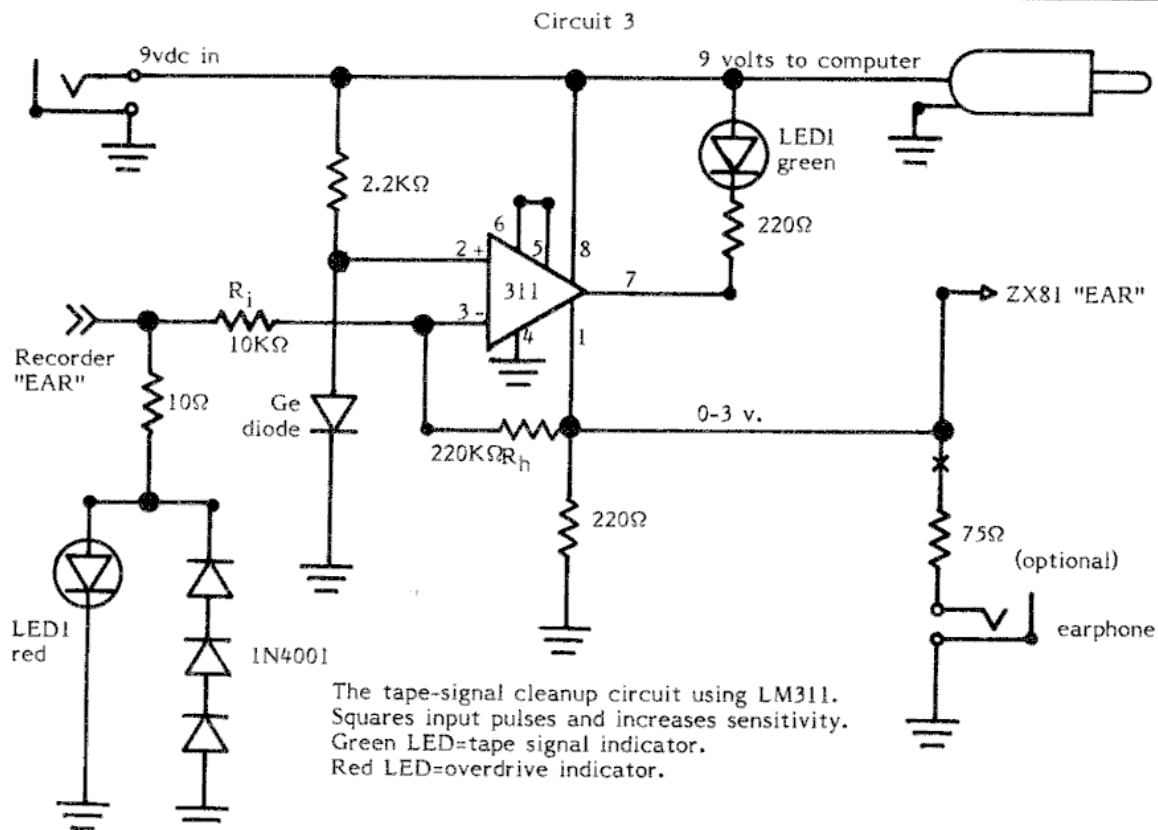




wider you make the hysteresis band (or "dead zone") with positive feedback, the less likely it is that the circuit will transmit glitches. On the other hand, too wide a hysteresis band may result in skipped pulses during a drop-out. The circuit shown in Circuit 3 is set up as a compromise, with the upper trip point at .3 volt and a hysteresis of about .1 volt. It uses a commonly available LM311 comparator, though many others can work; see the "Databooks" for suggested circuits. The circuit also includes a LOAD/SAVE switch and LED monitoring, and represents my "ultimate" load aid. A great many tapes that were balky before will be loadable now, and you can run your recorder at a lower volume setting.

If you're using a fast-load system like Z-XLR8, SDS, Q-SAVE, Fastloader, etc., you'll get better results if you add a .01 uF. capacitor in series with R_i (between RCDR "EAR" and R_i) and a 1K resistor from the junction of the cap and R_i to ground.

What if all this is too much trouble for you, and you'd rather just buy a loading aid? Which one is the "ultimate" in this case? Well, whichever one ultimately solves your problem. The least expensive option is G. Russell Electronics' "Winky Board II," which basically gives you visual and acoustic monitoring, a LOAD clipper and a SAVE filter to reduce RAM pack noise. It also allows you to patch two tape-recorders together for multiple saves or



direct copies. (Tape-to-tape copies are MUCH more prone to trouble than tapes saved from the computer, although the direct patch may at times prove useful.) The price is not unreasonable (\$19.95 A&T, 14.95 kit) and includes an in-depth manual. I have one here for testing, and it does everything advertised. I should mention that it appears to attenuate (reduce) the signal slightly, with the result that some extremely "thin" tapes won't load even at "full blast." However, you're better off without tapes like that around anyway. If you get an unusually quiet tape from a manufacturer or dealer, consider sending it back for replacement if reasonable efforts to get it loaded fail.

The accessory I personally recommend for a tape-cleanup device is the VOTEM. Unlike Winky, which is a passive device, VOTEM uses the computer's supply to power a comparator much like our Figure 4. I have found VOTEM's tape clean-up circuitry to be extremely effective; even on fast-load tapes where some other devices fail. (I do, however, suggest the addition of a high-pass filter before VOTEM when used with quick-load routines; i.e. .01 uF. in series, and 1K from VOTEM input to ground.) With the modification shown earlier this issue, the only minor "bug" (in the LED monitor circuit) is corrected. The result; even if data acquisition is not your favorite subject, your VOTEM will not sit about idly if you buy one. It's almost worth its price for the tape conditioner alone.

If after all this you STILL have trouble, face it, pal; you need a new tape recorder. Consider the Timex data recorder, which appears to be a virtual clone of the Minisette IX at about \$30 less cost. (Though after getting one and tearing into it, I see why it's cheaper; it's not quite as solid as the Minisette, but it is very good.) While I suppose there could be obscure computer problems that could cause save/load difficulties, it is far more likely that trouble will lie with the magnetic/ electronic/ mechanical contraptions we call "tape recorders."

If you have further data on any of this, or if there's something you feel we should have covered, drop me a line. We always can use more knowledge.

OFF THE WALL....

This entire article, including this blurb just fits into 16K of RAM using Memotext, with a little room to spare for editing. It takes 32 page flips to scan from start to finish. Just another answer to "How big is 16K?" 64K allows 3 times the storage on the ZX machines, or about 15 pages of SWN. So that means that the last issue of SWN was of the order of 160K in content, corresponding to about 1.3 million bits of binary information. Just thought you'd like to know.

FORTH: LANGUAGE UNIQUE

I'm sure that many of you have heard rumblings or seen ads for a new computer language called "FORTH," and are wondering what this is all about. The claims made for it are quite impressive; FORTH enthusiasts are almost evangelical in getting this tool known and used. Three versions (that I know of) exist for the ZX81/TS1000/1500 - "X-FORTH," "ZX-FORTH," and "TREE-FORTH." It is not our goal here to provide comparative reviews of these, as I'm a rank neophyte myself and wouldn't know how to use them, let alone compare them. Instead, we are presenting an article by Gary Smith of Hawg Wild Software. Hawg Wild is the distributor of "X-FORTH" for the T/S machines, as well as FORTH and other products for other machines as VIC, ATARI, etc. More will follow about Hawg Wild after the article, along with further info on how to get into FORTH.

Software is a notoriously elusive "substance;" whoever coined the phrase "nailing jelly to a tree" to describe computer programming had the right idea. Even BASIC programs can be quite "slippery," as anyone who has written such programs will attest. Assembly language programming ("machine code") is far more slippery. It is the job of any computer language or operating system to translate commands and operations from human concepts to the maddeningly simplistic commands the CPU acts on.

Result: high-level languages are easier to program, de-bug, and modify than "direct" or low-level assembly.

You probably already know the bad news. High-level languages (e.g. ZX81 BASIC) are terribly slow; the computer usually has to spend more time figuring out what you want to do, than actually doing it. Timex/Sinclair BASIC, like the other BASICs allowing direct commands, is "interpretive"; the translation to machine-code is done as the program runs, and each program line is first interpreted, then executed. A significant improvement is to use a compiler to first translate your BASIC code to machine-code (usually destroying the BASIC in the process). While being very useful for some jobs, compilers have serious drawbacks for others; the final (object) code is difficult to modify, you have to LOAD the original source code (BASIC), modify and re-compile. Also, BASIC programs usually have to be changed to suit the quirks of a particular compiler. There are none that I know of that include floating-point numbers, transcendental functions, and other nice features of Sinclair BASIC. Finally, while more compact than high-level BASIC, compiled programs generally take up more space than efficiently-coded assembly.

This is partially due to the fact that compiled BASIC is still structured like BASIC, i.e. in "human" terms. If a language were structured more along the lines of how the CPU "thinks," we could presumably write faster, shorter programs. If we could also define new operations or "words" as we go along, we'd REALLY have something. FORTH is such a language. You more or less "meet the computer half-way" and have to become familiar with how the stack works, using post-fix notation, and other concepts. In exchange, you are allowed unlimited flexibility in defining new operations. In a way, then, each program is written in a language exactly suited to that problem. You get assembly-level flexibility without its "slipperiness" and debugging hassles. 102 This article was originally published by Chet Lambert's Computer Trader (1704 Sam Drive, Birmingham, AL 35235) and is reprinted here with the author's permission. The Computer Trader is perhaps the first "grass-roots" computer publication, and certainly one of the more successful ones. It is definitely worth the \$15 per year (12 iss.) subs. rate, and supplies excellent info on a wide variety of computer topics. It's gratifying to note that recent issues have included more material on "our" machines, especially as regards hardware mods. Gary Smith is a frequent contributor, his column "Rnd Thoughts" contains frequent attention to ZX/TS machines.

Forth Origins

by Gary Smith
(C) 1983
HAWG WILD Software
PO Box 7668
Little Rock, AR 72217

Chet Lambert, the publisher/ editor/ copyboy/ etc. of COMPUTER TRADER has prompted this article. At first, I resisted because of my identification with FORTH, but relented because there is a need to share information.

So there is no misunderstanding, the home computerist's supply I head, Hawg Wild Software (TM), is heavily committed to FORTH. We have FORTH packages from several vendors for all home computers. It is, then, completely fair to believe a bias could flavor any article I might write about FORTH; you may consider that as you read this article. I will make every attempt to present the simple, unvarnished facts as I explain what FORTH is.

FORTH has been described as the only computer language NOT designed by a committee. Whether it is the only such language is debatable, but it is entirely true that FORTH is the direct result of one man's efforts to program more efficiently. Starting in the early sixties, Charles Moore, then a physics undergraduate at Massachusetts Institute of Technology and a staff member at the Smithsonian Astrophysical Observatory, began trying to develop a better programming tool. His intent was not to create a new language, but rather make himself a more productive programmer.

He had determined he could produce about one good, meaningful program in FORTRAN per year. In frustration he began experimenting with stack structures, post-fix notation (Reverse Polish Notation, or RPN) and the like. By doing so, he believed he could increase his productivity ten-fold. (Most programmers report MUCH higher increases in productivity with FORTH).

While at Mohasco Industries, in the late sixties, he dubbed his programming "tool" FORTH. The IBM 1130 he was working with was a third generation hardware machine (i.e. discreet components; diodes, transistors, resistors; on circuit boards), and he believed his "tool" was a software leap into the fourth generation. The IBM 1130 would accept only five-letter descriptors, hence "fourth" became "FORTH."

The first real test for FORTH was when it was implemented at Kitt's Peak National Observatory, AZ to guide the radio-telescope there. The results were so successful that calls began coming in from other astronomers anxious to duplicate the method. From there, word has spread largely by word-of-mouth, and by the efforts of the FORTH Interest Group (PO Box 1105, San Carlos, CA 94070; membership \$15.00 per year, includes monthly magazine, FORTH DIMENSIONS). So FORTH has grown from one man's conviction to what it is today, without the aid of a government or corporate push, just on the strengths of its merits and the sometimes fanatical enthusiasm of its adherents.

A word about that enthusiasm is called for here. Even FORTH's creator, Charles Moore, has said that FORTH polarizes users into camps of those who love it, and those who definitely do not. This is in part due to the language's uniqueness, and in part because it is something of an amplifier. That is, good FORTH programs tend to be very good, and bad ones tend to be lousy. I can say this; if you try FORTH and like it, you will very likely become a convert.

FORTH does not require or use the special registers, parameter passing, cache memory, or index addressing employed by large, mainframe computers to speed programming tasks. FORTH accomplishes much the same thing without all the attendant hardware just described. FORTH produces extremely compact programs; more compact than a non-FORTH user can usually believe. FORTH does not require special memory for error checking. Each word (function, item, operation...) may be tested as the program is developed. What all this translates to, is that FORTH is ideally suited to mini- and micro-computers, including (maybe especially) home computers.

Each computer's own resident binary machine code is the fastest running, most efficient method of programming that particular computer. As has been said, by me, in "Computer Trader," machine code is the only instruction set the machine truly understands. Next to true machine-code (often synonymous with) is assembler. Very close to these in terms of efficiency and execution speed is FORTH. No other high-level language consistently approaches FORTH in this department. There isn't even a contest.

Forth Numbers

As has already been said, FORTH is unique. Some people never adjust to FORTH's structure. That unique structure, though, is precisely what gives FORTH its efficiency and speed. If you use, or have used, a Hewlett-Packard brand of calculator, then you already know about Reverse Polish Notation. To add two numbers (5 and 3) in normal, in-fix or algebraic math (5 + 3) you place the operator between the two numbers. Not so with Reverse Polish Notation (origin country of the method's creator, Jan Ludasiewicz). With RPN, the operator follows the numbers (5 3 +). Note the spaces in between; they are an absolute must in FORTH.

The reason FORTH uses RPN is that it relies heavily on the parameter stack (called simply "the stack.") Most languages can and do use the stack, but its use is mostly user-transparent. With FORTH you MUST ALWAYS use the stack, and be aware of its contents. Likened to plates or trays in a cafeteria, the last number pushed on top of the stack is the first one popped off (i.e. LIFO, Last In / First Out). In our example above 5 is pushed on, 3 is pushed on; then during execution 3 is popped, 5 is popped, they are operated on (added), and the result (8) is pushed back on the stack. Compare that with the register moves for BASIC!

The WORD

Another unique feature is the "word." A word is the basic functional unit of FORTH. Most words manipulate the stack. "-" (minus), SWAP, EMIT (character to screen) are all words. The FORTH package comes with a dictionary of standard words (it must to call itself a standard package). You use this kernel to create your program. More importantly, you also use it to define new words to fit your application. These same new words will work on any other machine using FORTH. Furthermore, the new words are compiled in such a way that they are completely integrated into your FORTH system. These are two extremely powerful features: trans- portability and extensibility.

It must be pointed out that FORTH essentially has two versions, both highly transportable and for the most part compatible. In 1978 the FORTH Interest Group, convinced that FORTH would become popular if it were standard and available, released their fig-FORTH version. FORTH-79 is largely a subset of fig-FORTH intended to allow for greater transportability between systems. FORTH-79 is more applications oriented. The differences are generally minimal.

FORTH is structured. Not top-down, but bottom-up. FORTH is threaded, in that each new word has, not complete definitions, but pointers to its defining words. The programmer begins with simple conceptual words (at the bottom), and defines more and more complicated words (procedures) from them. The final, threaded word is the program. Any other new words created to fit the application are also part of "your" FORTH environment. In this manner FORTH continues to grow to fit the needs of the system, the application and the programmer.

This brings us to our closing point; the needs of the programmer. FORTH has been described as the software hacker's language. It has also been said that you can program FORTH to do anything, but you will probably have to program it yourself. You do have to do more with FORTH. You do have to keep track of the stack. You do have to guide FORTH to your application. FORTH, in turn, will do so much more for you. The key is that YOU ARE IN CONTROL. Each FORTH program is almost a new language; your language. (To that end, other languages have been written entirely in FORTH. It is used as an in-house development tool at several large corporations.) If you want a FORTH program to do more, YOU can extend it then and there.

CONTROL, STRUCTURE, EXTENSIBILITY, SPEED, EFFICIENCY, COMPACTNESS, TRANSPORTABILITY....FORTH

Best Regards, Gary
HAWG WILD Software (TM)

Questions & Answers

After receiving a sample of XFORTH from Hawg Wild Software, and looking it over for the first time, some questions came up which I put to Gary Smith. Since these are questions other first-time FORTH users are likely to ask, I'm including them here for all of you so that Gary doesn't have to repeat them.

- Q: XFORTH comes with a complete list of standard words and tape loading instructions, but does not provide much tutorial information. What would you recommend for first-time users?
- A: Our documentation is minimal; on purpose. You CAN NOT learn FORTH from abridged texts. We recommend "Starting FORTH" by Leo Brodie for tutorial purposes. We also supply a FORTH manual (\$14.95), written for Jupiter Ace 4000, but can be used as a general tutorial. "Starting FORTH" still best.
- Q: ("Loaded question" department) How does XFORTH compare with the other FORTH packages available for ZX81?
- A: Our FORTH is REAL, threaded FORTH and not arrayed (indirect thread). You get full FORTH restarts, etc. Read the ads. Ours is listed by FORTH Interest Group!
- Q: What kind of after-sale support do you provide?
- A: We publish XFORTH XCHANGE, on an "as needed" basis for XFORTH customers. We allow re-printing of any part of XFORTH XCHANGE by interested publications. This is good stuff for XFORTH users. No one else offers the support we do, at any price. (Gary's willingness to answer my dumb questions is proof enough for me! -- ed.)
- Q: The XFORTH package is about 6K long. Does all this get saved to tape each time, or can the object (result) code be SAVED independently?
- A: You SAVE both the Loader and Object. FORTH is threaded. Don't worry. Threading makes things VERY compact and fast. You can create very busy programs with 4K or less, let alone the 10K we leave!

Q: What is the Jupiter Ace 4000? How can it be obtained?

A: The Jupiter is currently the only micro with FORTH as its operating system. We (Hawg Wild) are authorized dealers for the Jupiter Ace 4000. Available now: (No longer being made. ed.) Jupiter Ace 4000 - \$150.00, 16K RAM pack - \$50.00, 48K RAM pack - \$125.00.

Hawg Wild Software sells XFORTH on cassette for \$25 + \$1 S&H.

Resources

MULTI-FORTH is available from Tree Systems, Suite 233, 3645 28th St., Grand Rapids, MI 49506. It is also EPROM based (it is the same as above and only available from Soft Magic. ed.), and comes with a card that the BASIC ROM plugs into, allowing switching between FORTH and BASIC. This is a multi-tasking version. Price \$49.95 ppd.

Another FORTH is called ZX-FORTH and comes from The Forth Dimension, 1451 Union St., Middletown, PA 17057. For \$42.95 you get the tape, "comprehensive user's manual," two sample programs (game and a simple WP), and a quick-reference card. Manual alone is \$10.00.

Soft Magic Corp., 1210 W. High St., Bryan, OH 43506. Their TREE-FORTH comes on a 64K EPROM, so you don't have to LOAD it from tape. Soft Magic is offering a special package deal (good only until final versions of docs and packaging complete); EPROM (reg. price \$69.95) plus "Starting FORTH" (retail value \$17.95) and "Understanding FORTH" by J Reymann (retail value 2.95) all for \$49.95. But hurry as this won't last for long.

Meanwhile, if this looks interesting to you, I would recommend that you purchase XFORTH and "Starting Forth" to get your feet wet. After gaining expertise you'll be in a better position to evaluate the merits of the other FORTHs for your applications. You should also contact FORTH Interest Group for additional "scoop."

SyncWare News is not affiliated with FIG, Hawg Wild Software, or any other FORTH supplier mentioned, nor are we receiving payment for any mention. The preceeding information has been presented for your info only, and we cannot be liable for inaccuracies. We ARE interested in your response and impressions (good or bad) re: FORTH in any of its versions.

Forth-with

By Gary Smith, (C)1984
POB 7668
Little Rock, AR 72217

Right now this column is little more than an idea that has been tossed about by myself and your Story Compiler, Fred Nachbaur. Whether it becomes a reality, and a continuing contribution to "SyncWare News" is strictly up to you, the reader.

I suppose before you pass judgement you should meet me and have a look at what I have to offer. So let me introduce myself. I am Gary Smith, proprietor of Hawg Wild Software (TM) and freelance writer.

Hawg Wild Software is a home computerist supply, specializing in, among other things, items for the Timex/Sinclair micros and the FORTH language (environment). We came into being by supplying 'XFORTH,' FORTH-79 for the ZX81/TS1000. As long as these two markets remain viable, we intend to continue serving them.

I have written articles and reviews for several magazines, as well as ongoing columns for "Computer Trader Magazine" (Lambert Publishing, 1704 Sam Drive, Birmingham, AL 35235), which will parallel, but not duplicate, this column if it receives your approval and thus continues.

I do not claim to be an expert on home computers or the FORTH language, but in fact far from it. I do have some knowledge of both, and if you wish I will be happy to share that knowledge with you.

What I plan to do in "FORTH-WITH" is present FORTH in small byte-size pieces, and in a way that should help introduce FORTH to the BASIC programmer; regardless of programming ability.

Since this is a Sinclair-specific magazine, I will also restrict examples to Timex, Sinclair and Jupiter machines. FORTH itself is the most transportable language in existence. There are, however, often times that FORTH words (the basic FORTH idiom, procedure, etc.) are enhanced if they are micro and even system specific. At those times, and in those examples I will make sure they apply to "our" machines.

I mentioned XFORTH for ZX/TS. Hawg Wild also has FORTH kernels available (CP/M, VIC-20, C-64, Spectrum) or under development (TS2068, [soon to be released] QL...) for most other home computers. If you wish to order a kernel, or be placed on our mailing list, or just chat please drop me a line at the above address.

If you would like to see this column continue, please let me know direct, or through Tom or Fred at SyncWare News. I would appreciate your comments on what you want and expect in the column.

[Editor's note: - The first of Gary's parallel FORTH articles appears in the June 1984 issue of the "Trader" (CTM). Have Chet Lambert start your subscription with this one. NOW THEN... Gary's comments are well worth heeding, not only as regards him, but also for other writers in this and other magazines. One sure way to blow away an author is to give him the feeling that he's writing into a vacuum. If you read an article you find interesting or useful, take a few moments to write to the author and tell him so! Realize also that the only reason you're getting "outside" authors in this mag is because they're willing to share information. If the apparency is that you couldn't care less, why should they continue to spend hours putting these articles together? YOU, the reader, are in control. If the author's address isn't given, write to SyncWare News and we'll forward your comments. -fn]

LPRINT HINTS

Getting Into A BIG Printer

I won't spend much time or space extolling the virtues of a "big" printer for your ZX/TS computer. Suffice it to say that a full-width printer can be the most useful investment you make since you first bought your computer. Such printers allow you to print full-width text and program listings in a variety of type styles and widths, on normal 8-1/2" paper. SWN is produced on a ZX81 with such a printer; I also use it to write all my letters, mailing labels, cassette labels, etc. (For this re-edit, and in fact for SWN Vol. 2, we have upgraded to a daisy-wheel printer instead of dot-matrix; the comments here in LPRINT HINTS apply for the most part to these as well.) Unfortunately there are some myths circulating to the effect that hooking up such a printer is difficult and tricky; in fact, it's not much rougher than using the TS2040, but there are some guidelines you should follow to get the most from your system.

First of course, you must decide on a printer. There are a good many decent dot-matrix printers available to do the job, from Epson, Okidata, Seikosha, Star Micronics, and others. Which one you choose depends on your budget, requirements, and tastes. Most manufacturers supply both an 80-column and a wider 132-column version; for most applications the 80-column type will be adequate. "Standard" features include double-width and compressed characters, an emphasize mode (each column of dots is struck, then moved half a dot and struck again, doubling the apparent resolution) and double-strike mode (each line printed twice.) Also, rapidly becoming "standard" on most machines are block graphics and "special" characters (more about this later).

An important specification is the character matrix. The traditional "cheap" printers (including the awful Gorilla Banana) use a 5x7 matrix, adequate for caps but with funny-looking lower case letters g, p, q, and y. More recently, competitive printers have a larger matrix (e.g. 9x9 for the Star Micronics "Gemini" on which the original SWN Vol. 1 was printed) which allows "true descenders" or lower-case letters whose tails actually drop below the line. Alternate character sets (e.g. italics) are available on some, as are a host of other features, some of which are nice to have and others you can do without. Look at the type of feed allowed - competitive units allow both tractor-feed (for perforated "computer paper") and friction-feed (for single sheets of bond, etc.)

But let's assume you've compared price and specs and have found a printer you like. You also

need a "printer interface" to connect the computer to the printer, as the T/S and the printer "don't speak the same language," and we need a "translator" to allow them to communicate. There are two types of interface, serial (RS232) and parallel ("Centronics,") in general use. Which system you'll need depends on the printer you choose. Older printers (like surplus bargains, etc.) will probably be serial, but most of the designs currently on the market use the Centronics-standard type of parallel interface (though most have a serial option available at additional cost). Recently, serial ports are once again preferred, as the same port or ports can be used to service printers, modems, and other I/O peripherals.

In either case, as far as the ZX/TS is concerned, the function of the interface is to, 1) control the operation of the printer from the computer, and 2) translate the data from "Sinclairese" to ASCII (the semi-standard code used by the printer). It does this with a combination of hardware and software (usually on PROM) and generally uses the uncommitted 8-16K region of memory. So now you can tell the computer what to do, which in turn, via the interface tells the printer, which executes and (via the interface) acknowledges completion.

The reason I referred to ASCII as being "semi-standard" is that while the codes from 0 to 127 are agreed upon, those from 128 to 255 are not. This is because originally most printers used only 7 bits, whereas modern machines generally employ all 8. As a result, what characters and control codes are placed in the high block is up the individual manufacturer, and may even vary from one model to another. This is where the block graphics and most of the special characters are. Interface manufacturers so far have solved the problem by avoiding it, though well-designed units leave you the option of accessing these codes if you supply the appropriate translation routine. The two "biggies" at the time this article is written are CAI and Memotech, though there are others and we would welcome your comments, good, bad, or indifferent, if you're using another brand.

I decided on the Memotech unit because it supports the T/S LPRINT, LLIST and COPY commands, whereas the CAI uses USR calls to perform printer functions. I find these USR calls more awkward to use, also they eat up more memory if your program calls the printer often. What I don't like about the Memotech is that some printer codes have to be accessed in a roundabout way. Overall though, I'm quite happy with it, so this series of articles will concentrate on how to put Memotech's CIF to work. Again, I'd love to hear from CAI (and other) users so we can balance things out some. An important

thing to remember is that in general, the various systems are not inter-compatible. (Boy do I know about this! More about that later.) For example, if you get the Memotech, you won't be able to use the CAI stringy-floppy at the same time. And so on. So decide on a system carefully as you'll pretty much be locked into it.

Some 64K Rams Won't Work

If you try to use Memotech's CIF printer interface with 64K RAMs by manufacturers other than Memotech, you can run into problems due to conflicts, as the CIF unit occupies memory in the same region (8-16K) as the RAM. Memotech RAMs use a small ROM to do the block-switching/decoding necessary to use the I/F; by using this proprietary approach they help lock you into their system. On some 64K RAMs, however, including JLO's and some others, it is quite possible to get the printer interface to work in harmony with the RAM.

First off, you'll have to disable the 10-12K block of RAM to keep the RAM from fighting the CIF PROM which occupies 10-10.5K. The simple gate circuit shown below does this by inhibiting MREQ NOT when this block is addressed. You can now read data in the PROM without RAM getting in the way.

By adding lines to A10 and A9 as well, connecting them through inverters to two of the unused inputs of the LS30, you can narrow the "shut-off" to 10-10.5K, leaving 10.5-12K free for other programming or data.

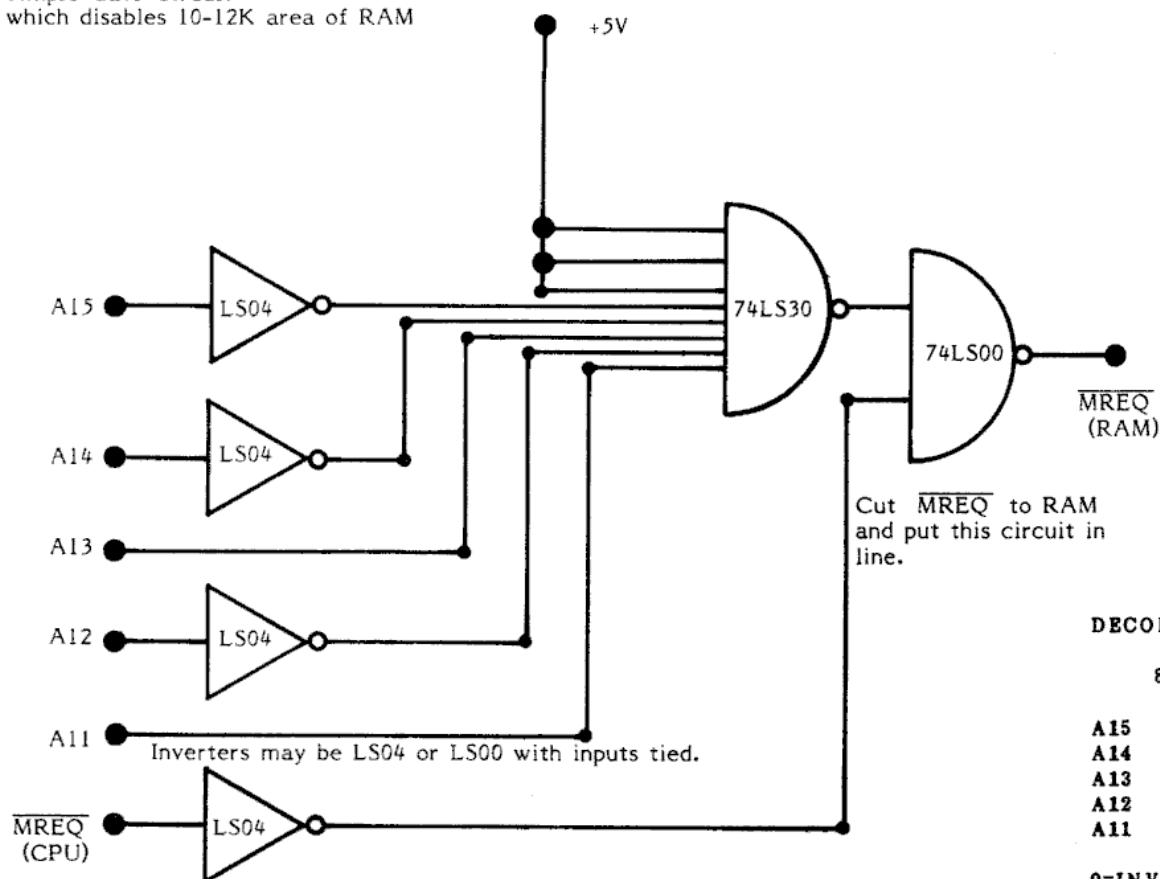
On units that use ROMCS NOT, you'll also have to diode OR the ROMCS NOT by placing a 1N914 diode in series, with the anode toward the RAM. (Thanks to John Oliger for this fix, which works fine on his 64K pack.)

Unfortunately, these simple fixes are not enough in the case of my little JRS 64K pack. Reports indicate that the same holds true for the Byte-Back UM64. These apparently use a different decoding scheme that inhibits proper CIF operation.

Memotech UK suggested linking edge connector pin 15A to pin 23A RFSH NOT. Doing this did not improve matters in my case. In addition this interfered with the proper operation of LDIR and LDDR into and from high memory.

If these suggestions don't help, and if you're not a consummate hardware wizard, you should probably consider getting another RAM (like JLO's) which does work with CIF. But if you want a challenge, here it is: can you find a way to make other high-mem RAMs work with this I/F? This would be a boon to people who already have someone else's RAM and would like to use the popular Memotech I/F. Hope my experiences have helped.

Simple Gate Circuit
which disables 10-12K area of RAM



DECODING OTHER BLOCKS

8-10 10-12 12-14 14-16

A15	0	0	0	0
A14	0	0	0	0
A13	1	1	1	1
A12	0	0	1	1
A11	0	1	0	1

0=INVERTED 1=DIRECT

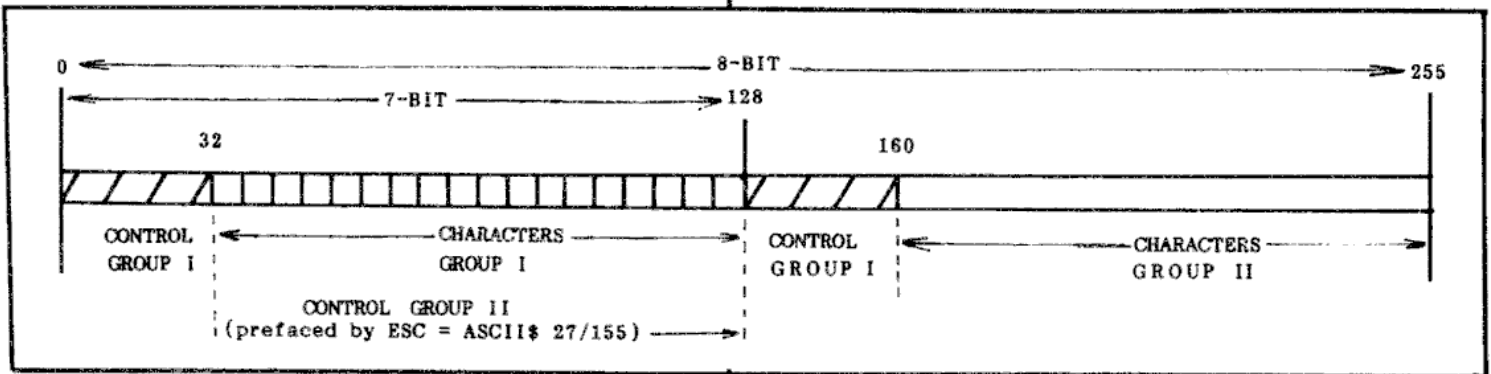
Controlling Your Printer

In this series of articles, I'll try to give you some information to help you use your printer system "to the max." We'll start with a discussion of the ASCII character set, and follow with some information about the Memotech Centronics Interface (CIF) which will demonstrate rudimentary methods of controlling the operation of your printer. In the next issue, we'll take what we know and develop some BASIC programs to do graphic screen dumps and write letters. You should then be able to adapt the programming to your system and needs.

Our goal throughout this series will be to complement rather than replace your ZX81, CIF (or SIF) and printer manuals; keep them handy as you will refer to them often. Our discussions will be

This leaves 95 codes for characters, plenty for upper & lower case letters, numerals, and symbols. But what's an equipment designer to do if the public clamors for even more available characters? Bright idea No. 1: use the eighth bit as well, for a total of 256 codes. Refer again to your printer manual for the characters that "live" in the high block; they will include block graphics, math symbols, etc. To insure 7-bit compatibility, the lower block remains the same; furthermore, the control codes are duplicated in the high block (128-159, 255). As a result, there are two codes for each control function in 8-bit systems; e.g. ESC can be called either with ASCII\$ 27 or ASCII\$ 155.

Great! Now we've got oodles of available characters (190 to be exact), but we still have only 33 control codes. Bright idea No. 2; assign a special control code (ESCAPE) that tells the printer that the next character is a control code. ASCII\$ 32-126 can then be used either "as-is" (characters) or to specify a control code (when prefaced by ESC = ASCII\$ 27/155). We thus have up to 128 control codes



general enough to apply to either the Centronics parallel (CIF) or RS232 Serial (SIF) interfaces by Memotech, though we will refer only to CIF.

First, a discussion of the ASCII character set is in order. This is the set of codes virtually all printers (and most computers) use for data transmission. Like the Sinclair character set, ASCII consists of 256 codes - but here the similarity ends. Compare the Sinclair set (Appendix A of ZX81 manual) with the ASCII table in your printer manual. They aren't even close! So it's quite apparent that translation is necessary; more about that later. To avoid confusion in this discussion, we will use CHR\$ to refer to Sinclair characters (only) and ASCII\$ to refer to ASCII characters. Printer manuals usually give examples using CHR\$. NOTE: By our definition these should be mentally changed to ASCII\$.

The ASCII code was originally designed for 7-bit interface, leaving the 8th bit free for future expansion. Thus, only the first 128 codes are really standardized. The first 32 codes (0-31) are not "characters" at all, but are various CONTROL CODES. These perform such functions as line feed, carriage return, change character width, etc. ASCII\$ 127 (DEL) is also a sort of control character (deletes last character sent). They have various mnemonic labels, but with the exception of ESCape and DElete these will probably not be of much use to you. Consult your printer manual for a description of each of these functions.

available. The diagram below should help clarify this.

The "usual" codes are shown (for our purposes) as "group I," and the "extended" codes as "group II." So how do we get Memotech's CIF to send these various groups? Let's start with the easy ones and work up.

Group I Characters

These characters are directly accessible using LPRINT. For example, LPRINT "X" and LPRINT CHR\$ 61 will both print an upper-case X. The CIF automatically translates CHR\$ into ASCII\$ using its built-in conversion table. Inverse letters are printed lower-case. Inverse symbols (except inverse period CHR\$ 155) are redefined as various useful symbols as #, %, !, @, etc. Three of the graphics symbols (CHR\$ 6, 136, and 137) are used for I/F control (HRG, etc.) and are off-limits. The other graphics are ignored (they're Group II characters, and are covered later). These "quirks" also exist using LLIST and COPY - which is why you should use CHR\$ (N) instead of "graphic" (e.g. PRINT CHR\$ 134, not PRINT "[graphic on Y]") if you want the program LLISTable. COPY is only useful for screen-dumping

alpha-numerics. As a side note, your printer is itself a sophisticated computing system, complete with CPU(s), RAM, and operating system, and yes it can be crashed like any computer. If you like crash-graphics you'll love some of the printouts you get when you do a no-no. Often the computer will stay up, and you just have to re-initialize printer using the ON-OFF switch.

Controls Group I

These are also easy to access. Along the lines of "Bright Idea No. 2," Memotech provides an easy access by defining an "escape code" (CHR\$ 155 - inverse period) that allows you to imbed control codes 0-32 in your text. CHR\$ 155 flags the INTERFACE that the following character is a control. The numbers and letters 0-9, A-V are automatically translated to ASCII\$ 0-31. (Subtracts 28 from Sinclair code in this special case.) Note that even though the same number is used as for the upper ASCII ESC code (a nice touch, perhaps serendipitous) the printer itself need not see this CHR\$ 155 as it doesn't require ESC to access the Group I controls.

Group II Controls - Characters

I'm lumping these together, since as far as the CIF is concerned they are handled identically. Since they are "untranslatable" by CIF, Memotech provided a "back door" with which you can send ASCII codes directly - including but not limited to the "standard" characters below 128. This is done by POKEing ASCII code into a 1 REM statement (which must be the first thing in your program) and then executing (send to printer) when you use the command LPRINT CHR\$ 6 (or LPRINT "[graphic on T]"). Everything from address 16514 on is sent to printer un-translated. By prefacing codes with ESC (good old ASCII\$ 27/155), we can send controls as well.

Since what you're putting into this 1 REM statement is in ASCII, when you LIST 1 your screen won't show anything near what will be printed. Some codes will goof up the LIST routine, so you should avoid listing it anyway. (By the way, if you're having trouble following this, refer to the CIF manual and the "Hello Badger" example.)

Conversion Table

Gee, why not make up a custom conversion table for your specific printer? Good idea. That will provide the basis for graphics dumps and math symbols, etc. We'll close this installment with an example of how to accomplish this. Play with it, figure out how it works, and see if that won't help spruce up your printed output! Next time you get my solution to graphic dumps and a "Caveman WP," but until then, why not work up your very own?

CONVERSION TABLE LOADER

```

1 REM XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
10 FOR A=16600 TO 16855
20 PRINT (STR$ (A-600)) (3 TO )
;"";
30 INPUT B
40 PRINT (STR$ (B+1000)) (2 TO
);"";
50 POKE A,B
60 IF A=16687 OR A=16775 THEN
GOTO 80
70 NEXT A
80 COPY
90 CLS
100 IF A<16856 THEN GOTO 70

```

Enter and RUN the conversion table loader. When you're prompted with a Sinclair code, enter the corresponding ASCII code from your printer manual. E.g., Sinclair code 00 = "space" = ASCII code 32, etc. If you have the printer on-line, it will COPY each screen when complete.

The following table, which I wrote for Gemini 10/10X, assigns special characters to the various Sinclair tokens.

You may have to change the codes over 160 to accommodate your machine. Then, overwrite the loader with the TEST program and add the 8000 subroutine that allows this fancy printing. RUN the TEST

TEST

```

10 LET Z$=""
20 FOR B=0 TO 255
30 LET Z$=Z$+CHR$ B
40 IF LEN Z$=64 THEN GOTO 70
50 NEXT B
60 IF B=257 THEN GOTO 100
70 GOSUB 8000
80 LET Z$=""
90 GOTO 50
100 PRINT "GRAPHIX TYPEWRITER"
110 PRINT "ENTER A STRING UP TO
60 CHAR."
120 INPUT Z$
130 IF LEN Z$>80 THEN LET Z$=Z$
( TO 80)
140 PRINT Z$
150 GOSUB 8000
160 GOTO 120
8000 REM GRAPHICS PRINT
8010 FOR A=16514 TO 16513+LEN Z$
8020 POKE A,PEEK (16600+CODE Z$(
A-16513))
8030 NEXT A
8040 POKE A,0
8050 LPRINT CHR$ 6
8060 RETURN

```

Conversion Table for Gemini 10/10X Dot Matrix Printer

000=032:001=225:002=227:003=231:	128=239:129=238:130=237:131=232:
004=226:005=233:006=230:007=235:	132=236:133=234:134=229:135=228:
008=174:009=252:010=251:011=034:	136=173:137=254:138=253:139=034:
012=195:013=036:014=058:015=063:	140=239:141=036:142=058:143=063:
016=040:017=041:018=062:019=060:	144=040:145=041:146=062:147=060:
020=061:021=043:022=045:023=042:	148=061:149=043:150=045:151=042:
024=047:025=059:026=039:027=046:	152=047:153=059:154=039:155=046:
028=048:029=049:030=050:031=051:	156=048:157=049:158=050:159=051:
032=052:033=053:034=054:035=055:	160=052:161=053:162=054:163=055:
036=056:037=057:038=065:039=066:	164=056:165=057:166=097:167=098:
040=067:041=068:042=069:043=070:	168=099:169=100:170=101:171=102:
044=071:045=072:046=073:047=074:	172=103:173=104:174=105:175=106:
048=075:049=076:050=077:051=078:	176=107:177=108:178=109:179=110:
052=079:053=080:054=081:055=082:	180=111:181=112:182=113:183=114:
056=083:057=084:058=085:059=086:	184=115:185=116:186=117:187=118:
060=087:061=088:062=089:063=090:	188=119:189=120:190=121:191=122:
064=182:065=188:066=187:067=032:	192=034:193=192:194=200:195=032:
068=032:069=032:070=032:071=032:	196=180:197=201:198=183:199=179:
072=032:073=032:074=032:075=032:	200=178:201=176:202=177:203=202:
076=032:077=032:078=032:079=032:	204=193:205=194:206=196:207=186:
080=032:081=032:082=032:083=032:	208=185:209=184:210=209:211=207:
084=032:085=032:086=032:087=032:	212=197:213=217:214=203:215=198:
088=032:089=032:090=032:091=032:	216=199:217=191:218=190:219=166:
092=032:093=032:094=032:095=032:	220=167:221=164:222=165:223=168:
096=032:097=032:098=032:099=032:	224=169:225=170:226=171:227=172:
100=032:101=032:102=032:103=032:	228=175:229=181:230=160:231=161:
104=032:105=032:106=032:107=032:	232=162:233=163:234=189:235=204:
108=032:109=032:110=032:111=032:	236=205:237=206:238=207:239=208:
112=032:113=032:114=032:115=032:	240=210:241=211:242=212:243=213:
116=032:117=032:118=032:119=032:	244=214:245=215:246=216:247=218:
120=032:121=032:122=032:123=032:	248=219:249=220:250=221:251=222:
124=032:125=032:126=032:127=032:	252=223:253=241:254=245:255=250:

program after SAVEing to tape. What you'll get is a self-test of your modified character set. Then you may enter strings to your heart's delight; graphics are a snap now, as are the "specials." Can you write a screen-dump subroutine? (Answer next issue!)

SPECIAL CHARACTERS

64	RND	□	212	USR	μ	234	REM	□
65	INKEY\$	±	213	STR\$	ρ	235	FOR	↳
66	PI	x	214	CHR\$	⊗	236	GOTO	↗
193	AT	⌘	215	NOT	¬	237	GOSUB	↘
194	TAB	↑	216	**	*	238	INPUT	Ⓜ
196	CODE	↳	217	OR	+	239	LOAD	Ⓜ
197	VAL	\$	218	AND	x	240	LIST	Ⓜ
198	LEN	Ⓜ	219	<=	+	241	LET	U
199	SIN	⊙	220	>=	+	242	PAUSE	Ⓜ
200	COS	⊕	221	<>	+	243	NEXT	Ⓜ
201	TAN	⌘	222	THEN	+	244	POKE	Ⓜ
202	ASN	λ	223	TO	Ⓜ	245	PRINT	Ⓜ
203	ACS	Ⓜ	224	STEP	Ⓜ	246	PLOT	U
204	ATN	Ⓜ	225	LPRINT	Ⓜ	247	RUN	Ⓜ
205	LN	Ⓜ	226	LLIST	Ⓜ	248	SAVE	Ⓜ
206	EXP	Ⓜ	227	STOP	Ⓜ	249	RAND	Ⓜ
207	INT	Ⓜ	228	SLOW	Ⓜ	250	IF	Ⓜ
208	SQR	Ⓜ	229	FAST	Ⓜ	251	CLS	Ⓜ
209	SGN	Ⓜ	230	NEW	Ⓜ	252	UNPLOT	Ⓜ
210	ABS	Ⓜ	231	SCROLL	Ⓜ	253	CLEAR	Ⓜ
211	PEEK	Ⓜ	232	CONT	Ⓜ	254	RETURN	Ⓜ
			233	DIM	Ⓜ	255	COPY	Ⓜ

Dump that Screen!

(For Memotech I/F Users)

If you took the trouble to enter an alternate conversion table as described in the last issue, you already have the most of what it takes to do a screen dump. The BASIC "DEMO" listing below, shows the required subroutine at lines 7000-7120. Whenever you want a copy of the screen printed out, GOSUB 7000 and the routine will look at each row in the display file and POKE that value's ASCII equivalent into the ASCII buffer at 16514. The last character (at 16546) is POKEd to zero as a terminator. LPRINT CHR\$ 6 sends the line to the printer.

First, in order to get graphics dumps, we have to change the line feed length (distance between lines vertically) as the graphics characters won't meet with the standard 1/6" line feed. To make each line exactly meet the next on Gemini-10, feed has to be changed to 1/12". This can be accomplished with the control sequence "ESC 3 n." This translates to 9B, 33, N (hex) or 155, 51, N (dec), and changes the line feed to N/144 inches. Taking what we learned in the last installment, it isn't difficult to come up with printer control commands to do this.

Take a look at line 8110. We're simply letting P\$ = the desired control sequence, in decimal. First, we send sequence 155, 35 (dec) or "ESC #" which makes Gemini 10 "accept eighth bit "as is" from host computer," which may not be necessary with

your machine or with the 10X. After this is the sequence 155, 51, 12 which sets line feed to 12/144ths or 1/12". Simply use CHR\$ XXX + ..., to concatenate the sequence into the string, as line 8110 shows, you can combine several control operations as long as you start each with 155 (9Bh) or 27 (1Bh) =ESC. Then, jump to 9000 with GOTO PC, and the values in the string are POKEd into the 16514 buffer. LPRINT CHR\$ 6 and the double CHR\$ 155 terminator (prevents spurious line feed) sends the codes and sets the printer to the desired mode. You can insert your own most often used control codes, then just GOSUB to actuate. Some examples are given which may or may not be agreeable with your machine, so check the manual. The CHR\$ 0 terminator after each sequence is really not necessary, as the routine at line 9040 does this for you. Some printer manuals (e.g. for Gemini 10X) don't give decimal values, so you'll have to translate from hex. Manuals are, of course, oriented around ASCII codes, so for example, to set 12 cpi the given sequence might be "ESC B 2." Somewhere though, you should find the hex (and sometime decimal) equivalents, i.e. 27 (or 155), 66, 2. Many other approaches are possible, including more sophisticated routines using variables like ESC=27 etc., so you can use the ASCII sequences directly. Our programming here is only one way of doing it, and you're certainly encouraged to experiment and improve.

Screen Dump Demo

(Line 1 as per previous Conversion Table Loader)

```

1 REM #M 4 STOP 44 LPRINT SC
ROLL LPRINT LPRINT SCROLL 4 LPRI
NT 44 LPRINT STOP LPRINT LPRINT
SCROLL LPRINT LPRINT 4 STOP 44 S
CROLL 4 SCROLL 4 XXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXX4 LPRINT STOP SCROLL LLIST
DIM NEW FOR # UNPLOT CLS 6780ZC
DYWXFHEJVBKLMNOPQRSTINKEY$PI???
????????????????????44444444
444444444444444444444444444444
444444444444444444444444444444
LOAD INPUT
GOSUB CONT GOTO REM FAST SLOW #
RETURN CLEAR 6 LOAD 8UCDYWXFHE
JVBKLMNOPQRST????????????????
??
10 LET PC=9000
20 SLOW
30 PRINT "ENHANCED? Y/N"
40 INPUT Y$
50 GOSUB 8250
60 IF Y$="Y" THEN GOSUB 8200
70 PRINT "DOUBLE-STRIKE? Y/N"
80 INPUT Y$
90 GOSUB 8450
100 IF Y$="Y" THEN GOSUB 8400
110 PRINT "PRESS ANY KEY TO S
TART DISPLAY. WHEN YOU WANT A SC
REEN DUMP, PRESS ANY KEY."
120 PAUSE 4E4
130 CLS
190 REM ZX MANDALA
200 GOSUB 400
210 PLOT 31+X,21+Y
220 PLOT 31+X,21-Y
230 PLOT 31-X,21+Y
240 PLOT 31-X,21-Y
250 GOSUB 400
260 UNPLOT 31+X,21+Y
270 UNPLOT 31+X,21-Y
280 UNPLOT 31-X,21+Y
290 UNPLOT 31-X,21-Y
300 IF INKEY$<>"" THEN GOSUB 70
00
310 GOTO 200
400 LET X=INT (RND*31)
410 LET Y=INT (RND*21)
420 RETURN
7000 REM GRAPHICS SCREEN DUMP
7005 GOSUB 8100
7010 LET DF=PEEK 16396+256*PEEK
16397

```

```

7020 FAST
7030 POKE 16546,0
7040 FOR H=0 TO 21
7050 FOR I=1 TO 32
7060 POKE 16513+I,PEEK (DF
+33*H+I)+16600)
7070 NEXT I
7080 LPRINT CHR$ 6
7090 NEXT H
7100 GOSUB 8300
7110 SLOW
7120 RETURN
8100 REM GRAPHICS SET-UP
8110 LET P$=CHR$ 155+CHR$ 35+CHR
$ 155+CHR$ 51+CHR$ 12+CHR$ 0
8120 GOTO PC
8200 REM ENHANCE
8210 LET P$=CHR$ 155+CHR$ 69+CHR
$ 0
8220 GOTO PC
8250 REM UN-ENHANCE
8260 LET P$=CHR$ 155+CHR$ 70+CHR
$ 0
8270 GOTO PC
8300 REM LF=1/6 (USE IF LEN OF S
TRING<=64)
8310 LET P$=CHR$ 155+CHR$ 50+CHR
$ 0
8320 GOTO PC
8350 REM LF=1/12 (USE IF LEN OF
STRING>64)
8360 LET P$=CHR$ 155+CHR$ 51+CHR
$ 12+CHR$ 0
8370 GOTO PC
8400 REM DOUBLE-STRIKE
8410 LET P$=CHR$ 155+CHR$ 71+CHR
$ 0
8420 GOTO PC
8450 REM SINGLE-STRIKE
8460 LET P$=CHR$ 155+CHR$ 72+CHR
$ 0
8470 GOTO PC
8500 REM ETCETERA
9000 REM PRINTER CONTROL
9010 FOR A=1 TO LEN P$
9020 POKE 16513+A,CODE P$(A)
9030 NEXT A
9040 POKE A+16513,0
9050 LPRINT CHR$ 6;CHR$ 155;CHR$
155
9060 RETURN

```

For dumping screens of text or numerical values, use COPY; if you try to mix text with our graphic screen dump the result will look "squished" as characters will run into each other vertically. Note also that printers don't have the grey and half-grey Sinclair characters, but you can modify your ASCII conversion table to print any desired printer character in their place. (E.g., the right-triangle graphics that is available on most printers.) You could also redefine the graphics as other characters for specialized printouts. POKEing the appropriate value into the conversion table is all it takes.

If you're into experimenting with machine code you might try to re-code the screen dump in assembly; it's not really vital, since the BASIC version is quick enough (in FAST mode) to outrun most printers. RUN the demo after making any printer-specific changes. As you see, it works just fine. (If it doesn't you might have a problem in your conversion table or printer-specific commands.) Experiment, try various possibilities and experiment more! You can only stand to learn.

A 'CAVEMAN' Wordprocessor

So, now that we can do screen dumps and engage the various printer control functions, how about a word processor? Sorry folks, but this is where your BASIC fails you utterly. It is just too slow for most people, especially if there is any input checking. So while there have been several valiant attempts at coding full-feature (or even not-so-full feature) word-processors in BASIC, all have the speed problem inherent in floating-point, interpretive high-level languages.

Now the good news. If all you need to do is write letters and other text that won't require extensive editing, and you aren't too annoyed by operating in FAST mode, then you can put together a workable "typewriter" in BASIC with a minimum of effort. Such a program is presented here. The program will print any selected number of columns, but it won't right-justify. It uses one long string to hold the text, and employs string slicing to change/add/delete words. You could extend it to edit larger blocks (such as sentences and paragraphs), but keep in mind that the more programming you add the less space you have for text. Another BASIC drawback; it's extravagant with memory. String slicing, while easy to implement, requires up to 3 times the length of the string during intermediate operations, so your practical limit is about 5K of text in 16K of memory, even with the most austere of BASIC drivers. However, it's easy, relatively fast and quite workable, if you're a reasonably good typist and don't have to do extensive editing. The first three issues of SWN (through 1:2) were written on a slight variation of this very program shown below! Though I now use Memotext exclusively, my "Caveman WP" served me well for correspondence and even newslettering for quite some time. I hope it does the same for some of you.

First, a few notes are prescribed. To avoid the double line-feed problem when printing strings over 64 characters long, the program adjusts the line

CAVEMAN

Line 1 as per Conversion Table

```

1 REM .. (AS PER LISTING FROM
SWN 1:3)
2 REM DONT DELETE
10 LET N0=0
11 LET N1=1
12 LET N2=2
14 LET N4=4
16 LET O0=10
20 LET T8=8000
22 LET H1=100
24 LET H4=N4+H1
35 LET P=N0
37 LET SC=704
38 LET PC=8900
45 LPRINT
50 GOSUB T8+H1
60 GOSUB 8300
100 LET Z$=""
105 GOSUB T8
110 LET Z$="F... .."
115 GOSUB T8
120 LET Z$="T... .."
125 GOSUB T8
130 LET Z$="E... .."
CO."
135 GOSUB T8
140 LPRINT
145 GOSUB 8200
150 LPRINT "E";"PO BOX 5177,"
155 LPRINT "E";"EL M... , CA."
91734
160 GOSUB T8+H4
165 IF P<>N0 THEN GOTO 187
170 PRINT "DATE=?";
175 INPUT D$
177 PRINT D$
178 PRINT "HEADING (H$)? 4 LINE
S"
180 DIM H$(N4,31)
182 FOR A=N1 TO N4
183 INPUT H$(A)
184 PRINT H$(A)
185 NEXT A
187 PRINT " LPRINT HEADING?"
188 PAUSE 40000
189 IF INKEY$<>"Y" THEN GOTO H4
190 LPRINT ",,D$"
191 LPRINT
192 LPRINT
193 FOR A=N1 TO N4
195 LPRINT H$(A)
197 NEXT A
198 IF P<>N0 THEN GOTO H4
200 CLS
205 PRINT "ENTER WRITER",,,,
BY F. NACHBAUR"
210 PRINT ",,NUMBER OF COLUMNS=
?"
220 INPUT Z
300 DIM A(N1+O0)
305 LET S$=""
310 FOR P=1 TO O0
315 CLS
317 PRINT "PARAGRAPH ";P
318 IF P=O0 THEN PRINT "LAST PA
RAGRAPH"
320 GOSUB 500
325 LET S$=S$+R$
330 LET A(P+N1)=LEN S$
340 IF A(P+N1)=A(P) THEN GOTO H
4-00
350 NEXT P
360 LET P=P-N1
400 CLS
401 PRINT "EXTENSION: PESS:
"
402 PRINT ",,""P"" TO PRINT TO
PAPER"
403 PRINT """S"" TO SAVE "
405 PRINT """R"" TO REVIEW/EDIT
"
406 PRINT """N"" TO START NEW L
ETTER"
409 PAUSE 40000
410 IF INKEY$="P" THEN GOTO 420
411 IF INKEY$="S" THEN SAVE "LE
TTER"
413 IF INKEY$="R" THEN GOSUB 70
0
414 IF INKEY$="N" THEN RUN
415 GOTO H4
420 FOR D=N1 TO P

```


When you've entered all your paragraphs, pressing ENTER (only) gets you out of "text creating" mode and goes on to the review/print menu. Press "R" to review; each paragraph is displayed in sequence. (If the paragraph is longer than one screen, then it is automatically split into two or more screens). Press any key (e.g. "R" or ENTER) to go on to the next screen or paragraph. If you find an error, press "E" (for edit). You are asked "which line;" count down how many lines (on the screen) the error is and enter the number. (If the error is in the second screen of a long paragraph, add 22 to the line number; e.g., the top line of the second screen would be 23, etc). At this point, you can also return to menu by entering "R" or to the next paragraph with ENTER only. Otherwise, you're now asked "which word?" Count how many spaces occur on that line before the incorrect word; enter this and the word is brought down to line 22. (If it starts right at the edge of the line, enter 0. Negative numbers are allowed, so you can even "space back" to a previous line if you goofed on the line number.) Now re-type it as it should be, complete with leading AND trailing space (or inverse space; we're using the I/F's conversion table here); and the offending word will be replaced with the correct version and the paragraph reprinted. If you omit the leading or trailing space, then the word will be run together with the preceding or following word. To delete the word, enter a single space. To leave it the way it was, press ENTER only. With a little practice and experimentation you'll quickly get the hang of it. The end of paragraph marker * is left in, but ignored in printing (actual limits are

```

8025 NEXT A
8030 POKE A,NO
8035 LPRINT "E="
8050 RETURN
8100 REM GR LF
8110 LET P$="70N"
8120 GOTO PC
8200 REM LF=1/6
8210 LET P$="M"
8220 GOTO PC
8250 REM LF=1/12
8260 LET P$="N"
8270 GOTO PC
8300 REM ENH
8310 LET P$=" "+CHR$ 69+" "
8320 GOTO PC
8400 REM UNEN
8410 LET P$=" "+CHR$ 70+" "
8420 GOTO PC
8500 REM 1 SHEET
8510 LET P$="S"
8520 GOTO PC
8550 REM FORM
8560 LET P$="T"
8570 GOTO PC
8900 FOR A=N1 TO LEN P$
8910 POKE 16513+A,CODE P$(A)
8920 NEXT A
8930 POKE A+16513,0
8950 LPRINT "E="
8960 RETURN

```

stored in array A, which is updated every time you make a change). Also, though the screen will show exactly as you typed and will make words wrap to the next line, your printed output will never split a word. See SWN through No. 2 for sample output. Build your own custom program from the pieces available here and elsewhere. You'll find that this kind of programming can be rather fun... though I wouldn't want to do it for a living!

MEMOTECH Oddities

(Re-edited to delete some "science-fiction," i.e. bugs.)

The Memotech WP "Memotext" is the best one I've seen for the ZX81/TS1000 to date (and as of Thanksgiving, 1984, still is-ed.); after owning one for some time I can't imagine life without it. Unfortunately, Memotech only provides the bare minimum of documentation and support needed to get your system working; there is little or no material available that delves into the fine points of the Memotech devices (specifically, CIF/SIF and Memotext). So here are some comments, observation and other info which might help you get the most out of the Sinclair/Memotech system.

First off, there's the hardware aspect of these units. Memotech took the liberty of taking over the operating system quite completely, and you can run into problems if you use other than Memotech 32-64K RAM packs, with their ROM-based decoding scheme. With 16K packs there is generally no problem, since there's no possibility of conflict in the 8-16K range. But even if your 32K-up pack allows you to turn off this range, some inexpensive 64K RAM packs still won't work; you might turn off the 8-16K range, install CIF and PEEK the 10-10.5K region to find the conversion table and IF program where it should be; but the printer simply won't print, or the system crashes. Your best bet in such cases may be to get a different 64K to replace it. If you're not interested in paying Memotech's price for their 64K, read on!

Now for some good news. The 64K RAM board by John Oliger (see review) works fine with the CIF/Mtext combo. The bad news is that, as assembled per instructions, the board won't work with CIF alone (useful for dumping older BASIC files, LList programs, graphics, etc.) The good news is that this is easily corrected on most machines by inserting a 1N914 diode in series with the ROM CS NOT line (pin 23B) to the RAM, anode end toward the RAM.

Another hardware oddity is the switch on Memotext. The manual implies that it must be ON for Memotext to work, but doesn't delve into much detail. Basically, when you flip the switch ON, the M/text program is engaged but you remain in BASIC until a key is depressed. Memotext then takes over and initializes the WP. Once in, there's no way to exit or return to BASIC.

A highly recommended addition to your 32K-up machine is to install a RESET switch (a momentary contact push-button, normally open, connected through a 100 ohm resistor between CPU pin 26 and ground, i.e., across capacitor C5). If you have a Hunter board you already have a reset switch (yet another reason to get one). Hit the reset button at the same time as you turn off Memotext, and you'll restart the computer (and clear the 16-32K range) while leaving the other ranges alone (i.e. where you keep your vital utilities, other files, or whatever).

This "auto-boot" feature makes it rough to disassemble the program if you're interested in such things. Tom suggested a way of doing this: Generate a REM statement as long as the length of code you want to examine, then write a loop to POKE the values from the 8-16K block into the REM; precede the loop with a PAUSE 300, and place a PAUSE 4E4 (pron. "forever") at the end. Run the program in FAST mode, and during the first PAUSE flip Memotext on; the loop will now work, since you haven't pressed a key since turning Mtext on. When done, the screen will go white again, (pausing 4E4) and you may turn M/text off, then BREAK to get into your program. You now have the desired code in your REM statement, whence you can SAVE it for later disassembly.

A much easier way is to simply LOAD HOT-Z II (16K "version"), turn Memotext on and wander about in the 2000h-4000h (8-16K) range to your heart's content. HOT Z II is such a "universal solvent" that it maintains control even over the built-in "Memotricks" in disassembly mode, but my meager efforts at single-stepping have left my machine in knots. When leaving HOT Z II (shift Q) you will return to BASIC if the switch is off, or to Memotext if it's on.

Memotext automatically appropriates all of available RAM, regardless of your initial RAMTOP setting. However, if you're careful not to fill up your RAM beyond, say, the 32K mark (16K of text), you can store other "goodies" (e.g. Hot Z) in high memory; when you hit the reset button to escape Memotext, the goodies will still be there except for the very top where Memotext places the machine stack. (In the original SWN 1:4 I incorrectly reported that M/text doesn't mess with RAMTOP; if any of you have had trouble as a result, so sorry. In the RAM-based version I've worked up, this oddity has been eliminated, and you can either use all of available RAM as supplied, or else make M/text respect your RAMTOP setting.)

Now, some comments on the software and hints on using it. One valuable tool sometimes overlooked by beginners is the "extended cursor movement" (fast scan) which flips through your text a screenful at a time. E.g., if you're looking at the first screenful of a long piece of text, shift"3" shift"6" shows the next screen, starting with the last line of the first screen for continuity.

When writing several different text files holding a control sequence in common, you can SAVE the control sequence only, then LOAD it whenever needed; rename it after loading, and you've saved yourself re-typing the sequence. Example: when originally writing this newsletter, an article usually started out by setting character width to 1/10", page width to 65 characters, line feed to 1/6", enhanced mode, and expanded characters for the headline. While I've gotten pretty good at typing these controls as needed, I could have saved some time by saving the most often used sequences, (e.g. "PICA", "ELITE", etc.) then load text file PICA and rename it "ARTICLE" and then amend text file "ARTICLE." The next file might be an editorial, so I'd load "ELITE" and rename it "EDITORIAL."

Now, a few words regarding speed. While it is indeed almost impossible to out-type when your current typing line is near the top of the screen, things slow down as the screen fills and takes longer to print each time. The faster you are as a

typist, the sooner you'll notice the slow-down as the screen fills. There is an easy fix. As you're typing along, when you get about 2/3 of the screen filled, hit shift"3" shift"6" (forward fast scan) and your current position is relocated at the top of the screen, and you may resume full-speed typing. If you wish to see the last couple lines of your text, shift"7" as required, then shift"6" (and shift"8") to get the cursor to the desired typing position. When inserting text during editing, use the cursor control keys to put the cursor near the top of the screen for optimum speed. Things get kind of slow in this mode anyway, particularly if there is a large body of text after the current cursor position. In such cases you need all the speed up you can get and will find it worth the trouble to figure out how to get the cursor to the top of the screen when typing.

I don't need to belabor the convenience of the other extended (prefaced by shift 3) commands; finding/exchanging/moving, blocks of text. The only problem is that it only works within a given file, i.e., you can't exchange text from another file. You can simulate some of these features with clever tape recorder use as described above, but in general you'll just have to grin and bear it. Another thing that would be nice is if data files could support control codes; the closest thing to this is the HINT on page 14 of the manual, using dummy strings for your controls and then using the exchange function to stick in the actual control sequence.

I got a couple requests for data on what areas of memory are used by CIF and Memotext. CIF uses the 10-10.5K area (10240 to 10751 or 2800h-29FFh). Memotext takes up 8-10K AND 12-16K (2000h-27FFh and 3000h-3FFFh). The 10.5-12K area is not used at all (not even for paging the hardware as previously suspected). The Sinclair-ASCII conversion table for CIF is at 2800h, the table for Memotext is at 3000h. (This is necessary because Memotext reverses the cases). ONE SMALL PROBLEM: when Memotext is used, the apostrophe ' is re-defined. When using the CIF table, the apostrophe is the inverse colon (graphic shift Z); but when using Memotext, the apostrophe is on inverse comma (shift Enter shift comma) and the inverse colon is redefined as the reverse apostrophe. As a result there were a lot of reversed apostrophes in issue 3, left over from before we found out about it (and broke our habits). Again, Memotech was of no help. The CIF manual even has a typo for this, showing the apostrophe as being on "inverse apostrophe" (Say again?)

For those of you who haven't figured out how to get the control commands (editU) to work, have no fear; it's really quite simple. Much like the "back door" in CIF, which allows us to send ASCII code in a 1 REM statement, the editU commands can be used to send characters or control codes. Example, to send a capital "X" use editU 58 editU (58h=88d=ASCII code for "X"). Similarly, editU 0E editU will send the control code for double width printing (SO). This takes care of the Group I controls and all available characters (see last issue). Group II controls are just as easy, just preface the command with 1Bh or 9Bh (27/155d = ESC). Example, to set 12 CPI on the Gemini use editU 1B 42 02 editU (ESC B 2 - same example used on page 12). One minor bug is that characters (but usually not controls) sent in this way screw up the justification by the number of such characters on the line. We haven't found a good way around this, the closest I've come is to add dummy characters (such as apostrophes) after the last word

on such lines, then use good old "wite-out" to remove them later. (A good way to accomplish this is to double type a letter in the same line and follow it with an editU 7F editU. Essentially, you are adding and deleting in hex, and the word processor never counts it. ed.) Also, you can't put more than 9 such characters in any one line; the system crashes if you do. If you have a lot of special characters on a line, the only way to send them successfully is to put the ENTIRE LINE in hex code between editU markers. This is a pain, but the only way I've found. (Thanks to Cedric Bastiaans, who ran into this problem and made me come up with a solution.)

Well, that's about all the space I can afford for this. Further discussion of the M/text system is reserved for Volume 2, where I show you how to run M/text in RAM to get rid of some of the annoyances.

Off the Wall

We'll leave you with this nonsense, inspired by my good buddy Tiger Williams. We feel it's time to de-throne "99 Bottles of Beer on the Wall" as the number one party tune, and came up with this to replace it. What do you think? (Mothers, tell your children; this is what can happen to a previously stable mind when becoming a ZX hacker....)

(To the tune of "Puttin' on the Ritz")

So you're blue
Don't know what to do
Machine stack is through
And cursor's gone too
Having you in fits
ZX on the fritz.

"Fritz" is "Fred,"
In German, it's said;
When power goes dead
From flashes of dread
Germans call it "Blitz,"
ZX on the Fritz.

(CHORUS)

It's no fun when programs go ker-blooeey,
But just be calm and find out what went screwy.
(Boo-boop-a-louie)

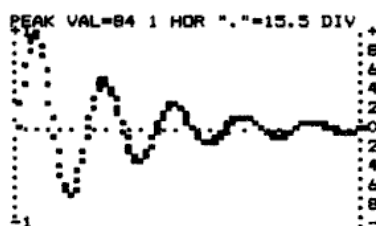
Typing all day
To go home and play
Two pages away
Your RAMpack will sway
Hits you like a blitz
ZX on the fritz

Midst curses and swears
You pull out your hairs
You're caught unawares
You'd rather fight bears
Than tangle with a glitch
ZX on the fritz.

Isn't it the pits?
ZX on the fritz.

(Chorus)

I'm sure you can think of other verses. Send them in, we'll print the best (worst?) we receive, so long as they're in reasonably good taste. (A little rhyme and meter doesn't hurt, either.)



TECHNICALLY
SPEAKING...

SOFTWARE NEWS

BRINGS YOU THE BEST !

BYTE PYNCHERS

A NIM Game for 2K

An important consideration in programming personal computers is memory economy. Unlike the big main-frames that offer virtually unlimited storage, you have only a limited RAM space at your ready disposal. Anyone who's tackled a large project knows just how small 16K can seem.

Fortunately, there are a few relatively "mechanical" techniques you can use to appreciably reduce the memory required by a program. Keeping your programming efficient is another step to getting the most out of (or rather into) your available RAM. We'll be running a "Byte-Pynchers" column on a semi-regular basis to show you some of these tips and techniques.

Meanwhile, here is a listing of a game which occupied about 4K of RAM as originally written, and was subsequently streamlined using several of these techniques. The result runs fine in an unexpanded 2K machine. The program will give you an idea of what

can be accomplished in limited memory if you give some care to using your RAM space as efficiently as possible.

If you're an experienced programmer, some of this will be old hat, but look it over! There's a possibility that there's something you can use. If you're a relative newcomer, some of this might look awkward and hard to read. This is the price we pay for byte-pynching. Future installments of "Byte-Pynchers" will hopefully help make sense of this stuff.

The game itself is quite simple; you and another player take turns removing as many "stones" as desired from any row. The point is to make your opponent take the last stone. It is OK to use RUN and CLEAR in this program.

If you're using an expanded machine (3.25 up), you should first fool the computer into thinking it has only 2K of memory by first entering POKE 16389,72 and then entering NEW. In 2K machines this is not necessary.

Lower-case letters indicate reverse video, except in lines 110 and 180 (see below.) As a last note, this is a game, unlike most, that pits you against another human instead of the computer's logic and randomizing function. The computer just provides the playing field and "referee" services. Have fun!

GRAPHICS NOTES:

Line 110: "gr. 5, gr.5, inv. *, space, inv. N,
inv. I, inv. M, space, inv. *, gr. 8, gr. 8"

Line 180: "gr. T, gr. Y, gr. T, *,
period, space" (change as desired.)

16K machines: To make the program run in 16K without changing RAMTOP (POKEing 16389) change line 320 to:

```
200 LET L=DF+66*H+N2*I+3
```

This takes into account the difference in the way the display file is handled with full memory.

```
10 SLOW
20 LET N0=VAL "0"
30 LET N1=VAL "1"
40 LET N2=VAL "2"
50 LET N7=VAL "7"
60 LET O7=VAL "17"
70 LET P1=VAL "21"
80 LET DF=PEEK 16396+256*PEEK
16397
90 LET T=N0
100 DIM N$(N2,N7)
110 PRINT AT N7,N7;"@@@ nim @@@
",,"BY F.NACHBAUR",,"
120 GOSUB VAL "250",,"
130 FOR A=N1 TO N2
140 PRINT "NAME-PLAYER ";A;"?"
150 INPUT N$(A)
170 NEXT A
180 LET B$="@@@*."
190 GOTO VAL "300"
200 LET L=DF+38*H+N2*I+N1
210 RETURN
220 INPUT A
230 IF A<N1 OR A>N7 THEN GOTO V
AL "220"
240 RETURN
250 FOR A=N0 TO N7*O7
255 IF INKEY$<>"" THEN RETURN
260 NEXT A
270 RETURN
300 CLS
```

```
340 LET X$="1111111"
350 LET N=N7*N7
360 PRINT "row",
370 FOR A=N1 TO N7
380 PRINT ,,, " ";A;" "; "o o o
o o o ";
390 NEXT A
500 PRINT AT O7,N0;"YOUR TURN,"
;N$(T+N1);AT P1,N0;"WHICH ROW?"
510 GOSUB 220
520 IF X$(A)="1" THEN GOTO 560
530 PRINT AT P1,N0;" row empty."
"
540 GOSUB VAL "250"
550 GOTO VAL "500"
560 LET H=A
570 PRINT AT P1,N0;"HOW MANY?"
"
580 GOSUB 220
590 LET S=A
600 FOR I=N7 TO N1 STEP -N1
610 GOSUB 200
620 IF PEEK L=N0 THEN NEXT I
630 IF S>I THEN GOTO 790
640 FOR I=1 TO I-S+N1 STEP -N1
650 GOSUB 200
660 FOR A=N1 TO LEN B$
670 POKE L,CODE B$(A)
680 NEXT A
720 NEXT I
730 IF I=N0 THEN LET X$(H)="0"
740 LET N=N-S
750 IF N=N0 THEN GOTO 820
760 IF N=N1 THEN GOTO 840
770 LET T=NOT T
780 GOTO 500
790 PRINT AT P1,N0;"too many."
"
800 GOSUB VAL "250"
810 GOTO 500
820 PRINT AT O7,N0;"YOU" "RE A F
OOL,";N$(T+N1)
830 GOTO VAL "850"
840 PRINT AT O7,N0;"YOU WIN, ";
N$(N1+T)
850 PRINT AT P1,N0;"AGAIN? Y/N"
860 IF INKEY$="Y" THEN GOTO 890
870 IF INKEY$="N" THEN STOP
880 GOTO 860
890 PRINT AT P1,N0;N$(N2);" STA
RTS?"
900 IF INKEY$<>"" THEN GOTO VAL
"900"
910 LET T=(INKEY$="Y")
930 IF INKEY$<>"Y" AND INKEY$<>
"N" THEN GOTO 910
940 GOTO VAL "300"
1000 SAVE "Nim"
1010 RUN
```

Numbers Are No No's

One of the big advantages of the T/S machine is also one of its disadvantages; its five-byte floating point binary numbers allow for 10-place decimal accuracy, but take more space to store than four-byte numbers. What's more, its BASIC uses floating point numbers, allowing calculated GOTO/GOSUB, FOR/NEXT in fractional steps, and other nifty things. But again, it's at the expense of memory space that we get these features.

Let's start out by counting the bytes in a "typical" program line, e.g. 10 PRINT 0. Well, the line number takes two bytes, length of line takes two bytes, and end of line marker (ENTER character) takes one. So it takes 5 bytes to define the line. The PRINT takes one byte and the 0 takes one, right? Wrong! The total length of this line is not 7 but 13 bytes -- almost twice as long as it looks! Enter the following program (ZX81/TS1000/1500) to see why:

```
10 PRINT 0
20 LIST
30 LET A=16509
40 PRINT A;" - ";PEEK A,CHR$ PEEK A
50 LET A=A+1
60 GOTO 40
```

RUN the program. You'll see the contents of the first program line displayed under the listing. Here I should note that most two-byte integers are stored (and used by CPU) "backwards" -- i.e. LESS significant byte first. The exception (there's always an exception) is the line numbers of BASIC programs, which is less significant byte LAST. So locations 16509 and 16510 give the line number as 0*256+10=10, and 16511 and 16512 give # of bytes as 9+256*0=9. 16513 contains the PRINT, and 16514 is the "0." But what's all that other garbage before the 118 end-of-line marker? Well, first there is a 126 that tells the LISTing routine that the previous expression (0 in this case) is a number, followed by the five-byte binary equivalent of that number. The bottom line is that every time you use a number in a program, you use up six bytes that don't show in the listing. Change line 10 and run the program after

each change. Try PRINT 1, LET C=PI, PRINT AT 0,0;0 etc. (This last one won't all fit on the screen; enter CONT to see as far into the program as you like.)

"What a waste!" you might well exclaim. But there's something you can do to reduce this wasted space -- avoid numbers in program listings whenever possible. "What? Compute without numbers?" Well, not exactly. What we need to avoid is FLOATING-POINT numbers in program listings. As you saw, it takes seven bytes to store even as "trivial" a number as zero or one.

One way is to refer to zero and one as NOT PI and NOT NOT PI, respectively. (Paul Holmgren suggested an even better one for 1 = SGN PI.) You could replace -1 with COS PI, I'm sure you can find others. Since most programs contain lots of zeros and ones, you can save a lot of memory by this technique.

Another way of avoiding floating point is to use VAL. E.g. GOTO VAL "1130" instead of GOTO 1130. This saves an automatic 3 bytes each use. This is great for one-time GOTOS as in menus, POKes, USR calls etc. but is appreciably slower; so, for inside loops and so on, you'll have to balance speed against economy when considering this.

Perhaps the most powerful technique is to use variables for the often-used numbers in your program. Change line 10 to PRINT B, enter LET B=0, then enter GOTO 10. Even though variable B represents a floating-point number, it takes only one byte to store "B" in the program listing. It takes six bytes to store this variable in the variables area, of course; but it's stored only once, so the more often you use B in the program

listing instead of 0, the more memory you save. For example, if you use 0 fifty times in the program (not all that much, really) you'll save close to 300 bytes by using a one-letter variable instead. Even if you add LET statements at the beginning of your program to define the constants, you'll still save memory if you use that constant more than 3 times. This has the advantage of making the program RUNable. If you're really squished for space, you can always delete the defining lines and remember to only use GOTO in the future, and to be as careful with RUN and CLEAR as you are with NEW.

Using up one-letter variables for constants is not recommended for larger programs, where you need them for your various true variables and FOR/NEXT variables. For scrunching a large program, a better approach is to use variables like N0=0, N1=1, ... , O0=10, ... P0=20, P1=21, etc. as needed. Commonly used subroutine numbers can be replaced with S1, S2, etc. While your savings won't be as great with two-byte names, they are still quite appreciable. For numbers not used often enough to merit their own variable but which are expressible as a simple function of existing constants, you can save a few bytes by using an expression. E.g. if O0=10 and P0=20, then we can say GOTO O0*P0 instead of GOTO 200 and save four bytes (and still run a good deal faster than using VAL.) The N?, O?, P? decade approach is almost as easy to read "real" numbers, and it's not an unconfontable task to go through an existing program and edit the letters before each "target" number.

Experiment with this using any of the available "bytes remaining" routines to measure your progress in economizing memory. You might be surprised how much you can stretch your RAM with these approaches. Until next time, happy pyningh!

Of Flags and Strings and Other Things

Well, folks, we were scooped by James Grosjean in SYNC magazine on the planned subject for this issue. So I'll refer you to SYNC 3:5 page 80 for all kinds of nifty ways (some of which I hadn't thought of) to save memory in your programs. We can thus go on to the next step in saving memory once you've used the standard bag of tricks. This is improving the memory-efficiency of entire routines. These will be things that become more useful the bigger and more complex your program is.

If you wrote out a program from start to finish, linearly, I guarantee it will contain redundant code. We all know that if you use ANY code more than once, you should place it in a subroutine. Then just GOSUB whenever you need that routine. But what about sections of code that are not identical but similar? Or what if there is a section of code in our main routine that we wish to access from a subroutine (sounds odd, but sometimes can prove quite useful)? These and similar things can be accomplished easily using logical operators and FLAGS. Flags are simply indicators of some condition. Your computer has some special FLAGS registers that allow conditional statements and generally keep track of things; we're not talking about these; our flags will be BASIC variables used to control operation of your BASIC program. Clever use of flags can save big chunks of memory and help organize your program. Tom's series uses flags extensively; if you understand his program you already know most of this.

An example could be the following routine that functions either as a prompted input or annotated print routine. N\$(1,N,M) would hold the variable names, (e.g. VCC, PMAX, BETA) and N\$(2,N,M) could be the appropriate units (VOLTS, WATTS, A./A.) You could fill this array manually and save the defining lines. Array A(N) holds the data. N= # of entries (up to 20), M= # of characters in labels (up to 9). F is our flag, when F = 1 it's an input routine, and when F = 0 it's a print routine. The section of code is effectively changed by the flag.

```
1000 CLS
1010 SCROLL
1020 IF NOT F THEN PRINT TAB 8;"TABLE OF VALUES"
1030 IF F THEN PRINT TAB 3;"INPUT VALUES IN UNITS SHOWN"
1040 SCROLL
1050 IF F THEN DIM A(N)
1060 FOR A=1 TO N
1070 SCROLL
1080 PRINT N$(1,A);TAB M+1;"=";TAB 12;("?" AND F);TAB 22;N$(2,A)
1090 IF F THEN INPUT A(A)
1100 PRINT AT 21,12;A(A)
1110 NEXT A
1120 PAUSE 4E4
1130 IF INKEY$="Z" THEN COPY
1140 RETURN
```

Play with this, I'm sure you can find some use for it. Just for fun, figure out how much memory is saved as opposed to writing it as two routines. For an interesting effect in "print" mode (F = 0), delete the AND F and brackets in line 1080. Notice how we can turn things on and off with the flag and logical operators? For a real challenge, chase down all of Tom's flags to get an idea how to use sections of code efficiently (hint- start at line 1000). To use a section alternately as a main routine or a subroutine, just put IF F THEN RETURN at the end; set F to 1, GOSUB (start), and it works just like a subroutine. If F=0 the RETURN is ineffective.

If the flag is referred to more than a few times, it is more efficient to use a numeric variable than a string, even though storing a one-character string takes five less bytes. This is because of the compactness of such statements as IF F THEN... and GOTO (F AND ...) + ... (a remarkably useful tool, by the way, especially for those memory-hog menus). But there's no reason why you can't use a string as a flag. Strings are especially useful if more than one choice is involved, as it takes more memory to say IF A=2 THEN... than it does to say IF A\$="2" THEN.... Also it depends on your preference for indicating choices; if you prefer the more deliberate approach of inputting a number (as Tom does), you're better off using a number for your director flag. But if you prefer the faster INKEY\$ approach (as I do) you are better off using a stringy flag.

Speaking of strings, that's our next topic. It would be nice if we could have one-byte integer arrays for such things as screen positions (PLOT or PRINT AT), integer counting, maps, etc. Well as a matter of fact, we have such a thing, but we need \$'s to access it. No, not some expensive peripheral, but your ZX81's excellent string (pron. "\$") handling ability. Most number-grinder type programmers are a little leery of strings, as it sounds suspiciously like data or word processing or something equally boring. But in fact, it's a great way to store positive one-byte integers. Assigning is easy; to change a number to a string, use CHR\$; to convert back to a number, use CODE. Review the manual sections if you're hazy on these, they're a worthy part of your BASIC vocabulary.

EXERCISE: (You said you wanted tutorials, and those have exercises.) Convert the F and G arrays in the SE Fit-plot routine into string arrays (E\$ and G\$). On a 40-point plot we could thus save about 400 bytes. HINTS: Line 5690 becomes DIM F\$(PD+A), line 5830 LET F\$(PF)=CHR\$ TX (no need for INT). Time the difference in PLOTTing time between the original and your modification. Was it worth it?

By the way, there's a good reason to do the exercise; line 5690 & 5700 is where you're likely to bomb in 16K if you specify a high fit density. Converting this to string arrays gives you a lot more headroom here.

I'll leave you with a treat; the first 11 lines of CE AMP which are two simple but very useful subroutines if you're dealing with large numbers of inter-related system parameters (e.g. all the quantities in an electric circuit). The input/print routine above shows one way of doing this; put the values in an array. Easy to input and print using a loop, but hard to follow in calculation lines if the various relationships are arbitrary. Much easier to use named numeric variables, like VCC, R1, IMAX, etc. so we can say things like LET IMAX=VCC/R1 and have the formulas easy to debug. Also, should we wish to read, say VCC, in immediate mode, we can just enter PRINT VCC instead of looking up what Vcc's array position is. But all those INPUT and PRINT statements! These routines get around this. Put a 4-character NAME into Z\$, and a numeric VALUE into C, and call subroutine 10; it defines a numeric variable named (contents of Z\$) and loads it with C. Use this for inputting or assigning values from an array. Going the other way, subroutine 20 looks up the variable named (contents of Z\$) into C. This is useful for printing. Enter the lines exactly as shown, don't change anything before line 25 unless you modify the POKE addresses (yes, the program changes itself. See Bill Yotter's ALGEBRA for another example of self-changing code, one that CAN be moved around. Look up the system variable at 16425... Now that wasn't so scary, was it?) Start your program at line 100, define N1=1 and N4=4 (constants; see last issue). Example might be variables in a kinetics problem, e.g. a space craft. Hope you have a good time with this!

```

5 GOTO 100
10 FOR B=N1 TO N4
11 POKE 16547+B,CODE Z$(B)
12 NEXT B
13 LET XXXX=C
14 RETURN
20 FOR B=N1 TO N4
21 POKE 16640+B,CODE Z$(B)
22 NEXT B
23 LET C=XXXX
24 RETURN

```

```

100 REM INPUT
110 LET N1=1
120 LET N4=4
130 LET R$="MASSDISTVIX ACCXV1Y ACCY"
140 FOR A=N1 TO LEN R$/N4
150 LET Z$=R$(A*N4-3 TO A*N4)
160 PRINT Z$,
170 INPUT C
180 PRINT C
190 GOSUB 10
200 NEXT A
210 STOP

```

By the way, self modifying code is definitely TABOO in structured programming. Just try to debug it sometime. ed.

HARDWARE PROJECTS

When it comes right down to it, there's not much the ZX/TS computers can be criticized for. The commonest complaints about the machine are that, by its construction, it is quite prone to crashing (often at a very wrong time) as a result of physical movement. RAMpacks are a prime candidate, or anything else on the rear edge-connector. Most people, after they've had their machine for a while, devise or purchase some method of making the peripheral port connection secure. But this still leaves another major source of "hardware crashes"--the power connection. The little mini-plug itself can be accidentally pulled, or a temporary power outage or the power supply getting knocked out can cause the same effect.

In The Custom T/S, we will investigate various simple modifications you can make to your machine which will make it virtually crashproof. Both of my machines are on all the time and simply don't crash for hardware reasons. Even if you're storing important routines in PROM or non-volatile RAM, you still need to secure power delivery for the CPU and volatile RAM.

Another common complaint is overheating (and improper operation) in warm weather. Common syndromes are dead keyboard sections, display tearing or other irregularities or (in severe cases) "clear-screenitis." Almost all cases are cured by a simple mod which won't put any holes in the case, etc.

These and other "problem-solvers" are covered in future issues. Here is an example of a more advanced but still quite simple project - putting a joystick connection to your computer. With this scheme, you can select which keys will be activated simply by how you connect it. I hope this will give you an idea of the possibilities you can get into if you're willing to experiment. You have the ideal computer to customize exactly to your requirements and tastes. The Custom T/S will show you some options.

Software Compatible Joystick

It's a snap to connect an Atari-type joystick to the ZX81/TS1000; all it takes is direct connection to the keyboard lines. The problem with this is that only certain combinations are possible, and you cannot hook it up so that the joystick controls the "arrow" keys 5-8; as a result, software written for the computer, which usually uses the arrow keys, has to be modified for use with the joystick. This is especially rough if the keyboard-sensing is done in machine-code, but is a hassle even to change BASIC INKEY\$ commands. So here is a simple (and CHEAP - about \$2) way to install a joystick that activates the arrow keys, or any other keys you wish.

There are two sets of lines that go to the keyboard; the KBD lines (5, numbered 0 to 4) and the DIODE lines (8, numbered 1 to 8.) Connecting one of the KBD lines to one of the DIODE lines produces a character unique to those lines, as shown in the table of Figure 1 below. For example, shorting KBD0 to D2 gives "Q," etc.

FIGURE 1

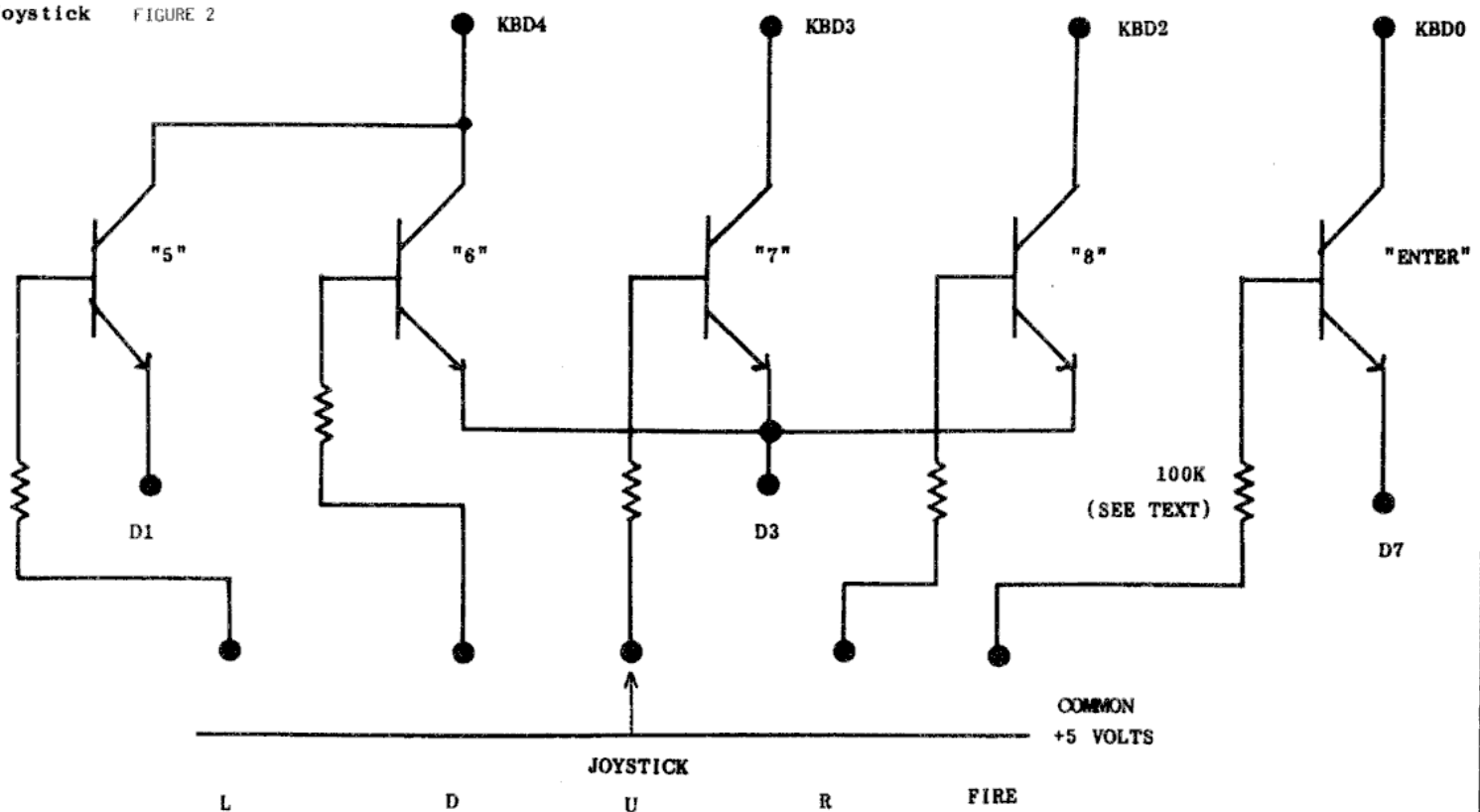
UNDERSIDE OF THE ZX81 CIRCUIT BOARD												
DIODE LINES								KBD LINES				
0	0	0	0	0	0	0	0	0	0	0	0	0
8	7	6	5	4	3	2	1	0	1	2	3	4
#	@	%	P	A	O	Q	1					
.	L	Z	O	S	9	W	2					
M	K	X	I	D	8	E	3					
N	J	C	U	F	7	R	4					
B	H	V	Y	G	6	T	5					

"#"= SPACE "@ "= ENTER "%"= SHIFT

Atari and similar joysticks use six lines; 4 for the directions, 1 for the fire button and 1 common return. So, you could connect the common to one of the KBD lines (e.g. KBD4) and the other five lines to the diode lines (e.g. 1-5.) In this case then, the joystick will activate 5, T, 6, G, and Y. If you study Fig. 1, you'll soon see that there's no way to control 5-8 with this approach.

The circuit of Fig. 2 overcomes this by using garden-variety transistors to do the actual switching; the connection shown puts the joystick directions on 5-8, and the fire button on ENTER, but any other combination is possible. Just tie the emitter of the transistor to the desired DIODE line, and the collector to the desired KBD line. Almost any low-power NPN silicon transistor will work, like 2N3904, 2N4401, 2N2222, etc. You might have to experiment with the values of the resistors. Use 100K as a starting point. If nothing happens when you activate a certain direction of the stick, increase the value. If the cursor flashes but no character is generated, the resistor value is too high. On my machine, all values are 100K except for 7 (up,) which is 33K for correct operation. You only have to remove the bottom of the computer case to "hard-wire" the stick. If you'll be installing a

Joystick FIGURE 2



socket, you may have to remove the board all the way. You can glue the transistors, flat-side down, to a 1/4" x 1" strip of plastic, which in turn glues to the PC board near the KBD connectors. Once you've determined the right values for the resistors, they can be glued to the valleys between the transistors. The five lines from the joystick go to the free ends of the resistors, and the +5V for the common is available at the COMMON connection to resistor pack RP3.

Joyful sticking!

The Custom T/S

Let's start our series on T/S reliability improvement with some simple but effective ways to make your basic machine virtually crashproof. If you've been using your machine for a while you've probably had times when the computer suddenly dies and nothing short of starting over will bring it back. I won't belabor how annoying this can be, suffice it to say that with only a few hours tinkering you can make your journey through computer-land a lot less frustrating. We'll concern ourselves with projects that are easily do-able by anyone with the simple mechanical skills with hand tools and a soldering iron; our first two projects, in fact, don't even involve soldering, yet can upgrade your system's reliability considerably. So you don't have to be a hardware whiz to customize your TS1000/ZX81, though of course at least some kit-building experience is recommended.

The most common (and most publicized) cause of hardware crashes is the "wobble & die" RAMpack-wiggle syndrome. As a result there have appeared a great many fixes for this, and we will assume that you've already secured this source of trouble one way or another. If you just recently got your machine, there are a number of products available to cure this, including straps, formed lucite supports, etc. The point behind it is in all cases to prevent the RAM from moving relative to the computer, which causes momentary bad connections to the edge-connector. This way of connecting to PC boards was designed and works great for cards that are mechanically secured and aren't moved except for service. Reliability suffers, though, if this type of connection is moved even a little bit due to the play between computer and RAM board. This simplest fix is to use "sticky foam" or velcro to semi-permanently secure the RAM case to the computer case. With sticky foam it may take more than one layer, and don't be too chintzy with it or you won't get the strength you need. Or you can get fancier with brackets or entire workstation layouts. Since the exact cure depends on whose RAM you have as well as your tastes and abilities, we'll leave this up to you and go on to secure the next source of hardware crashes - the all-important power connection, and while we're at it, the cassette plugs.

Plug Guard

You may have noticed that the power connection to the T/S machines is somewhat flimsy and prone to sudden death if it gets pulled by a tug on the cord or in some cases, even if it gets bumped the wrong way. So here's a way to guard against this possibility. Figure 1 shows a way to do this using a 1-1/8" x 2-1/8" piece of lucite or aluminum, 1/4-3/8" thick. Three 3/8" holes are drilled to match the spacing of the power and cassette plugs, and three perpendicular tapped holes allow for machine screws to lock each plug in its respective hole. Even if you don't attach the block to the computer, the three plugs are locked together as one assembly, making them much more reliable. Or you can drill two small holes as shown to screw the block to the upper computer case-half for the ultimate in security; the plugs can still be removed easily by loosening the set-screws, but when they're in and locked you can be sure they'll stay in under reasonable jostling.

Making the block won't be a big deal for you if you have machine-shop access and experience. If you're using lucite, you should have a circular saw and a lucite blade, and some experience or willingness to experiment with lucite; it binds easily so take it slow. Also take your time drilling, whether you use lucite or aluminum. You need a drill press and a good vise; a hand drill won't be accurate enough. Mark the holes carefully first then drill a hole with a 1/16" bit. Work up to 3/8" a size or two at a time to preserve accuracy as much as possible. If the holes end up incorrectly spaced or askew, the plugs won't fit properly and you'll damage the plastic jacks if you force them.

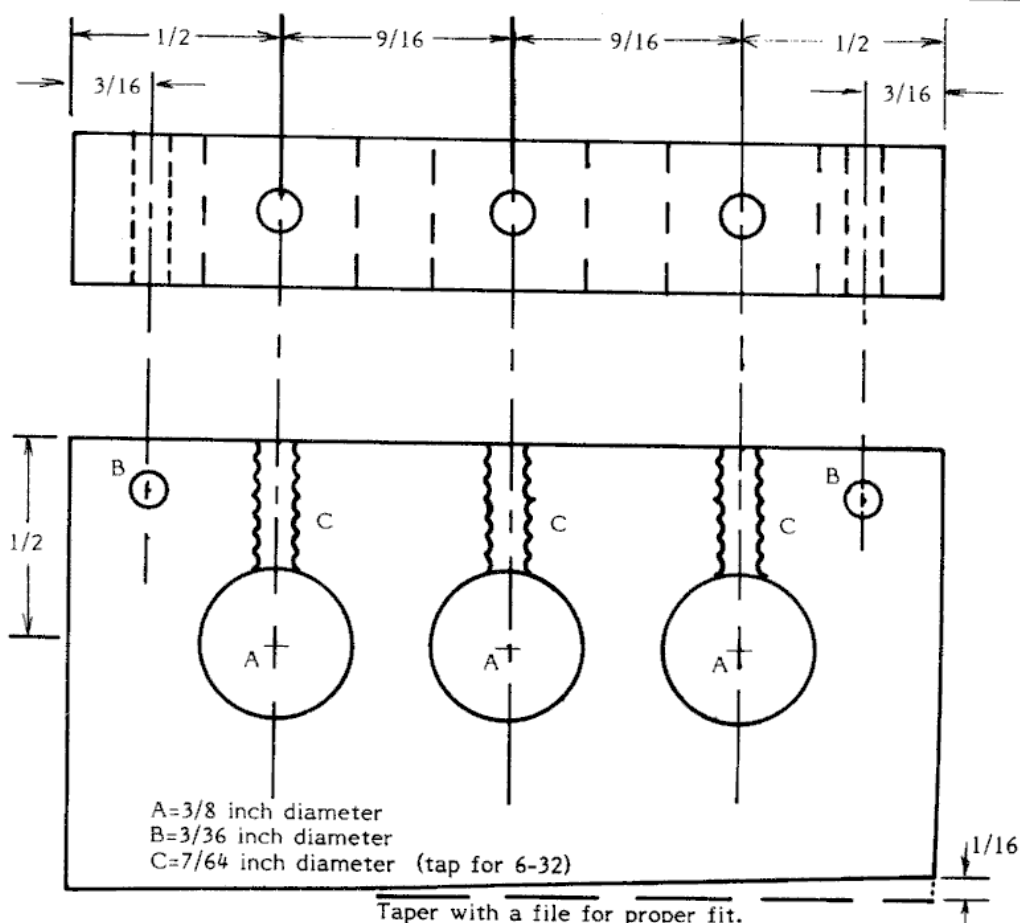
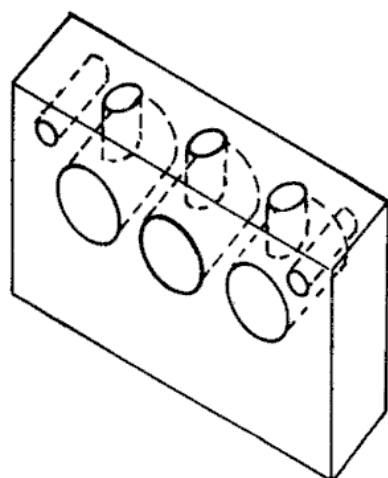
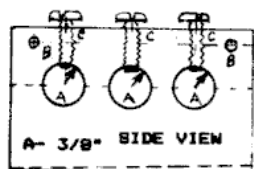
The dimensions shown work with the cords that came with my machines, but it's conceivable that it won't be right for yours, as these were apparently obtained by Timex/Sinclair from various suppliers. If this is the case, your best bet is to change the plugs to standard plugs like Switchcraft 750, available at virtually every electronics parts jobber. (Radio Shack also supplies a "clone" of this part.) These also fit more snugly than the molded type and are repairable if they develop a loose connection.

If this sounds like it's beyond your means, don't despair. Quite probably you know someone who could help you out if you ask nicely and maybe supply a six-pack. You might check out the small machine shops in your area; you may find someone to help you out for a token charge. If you have a lucite specialist in your area, he'll probably cut you the block from scrap and even do the holes at a bargain rate if he's not too busy. (And as long as you don't make a pest of yourself!)

If you want to permanently mount the plug guard to your computer, include the mounting holes on the guard and use these to mark where the holes should be on the computer with the plugs in place. CAREFULLY drill small (1/16") holes in the case and attach the guard with two, #2 x 1/2" self tapping screws. Do the final tightening with the plugs in place. You may have to file out the plug holes a little to allow easy insertion and removal, depending on how accurately you drilled them.

Should you be in a position to manufacture a number of these, let me know if you'd be able to make them for others who may not be willing or able to do it themselves.

Plug Guard



The Custom Cooler

It's sad but true that the ZX81/TS1000 machines tend to be sensitive to overheating in the summer. With a solderless mod, you can in most cases cure the "summertime blues." No, there's no drilling and carving involved either.

Most overheating problems can be attributed to the "workhorse" chip, or ULA (Uncommitted Logic Array) that co-ordinates the CPU and memory with TV monitor and cassette jacks. Less often, the CPU develops oddities (especially the SGS unit used in the later Timex's), and on occasion the 5V regulator is to blame. Generally, the regulator is quite capable of holding its own, unless the bolt to the heatsink is loose or you have a large peripherals current demand.

Before we go on, let's look at the cause of overheating and possibly bury some myths about "keeping cool." Heat is generated primarily in the silicon chips that make up the computer; how much heat depends on the power each chip consumes. Typically, the regulator "burns" 2.5 watts (with 16K RAM), the ULA 1.3 watts, the CPU 1.1 watts and the built-in RAM and ROM together only about .1 watt. How hot each chip gets (temperature) depends on its ability to get rid of heat to the surroundings. Heat developed by the chips has to pass through the IC package, where convection inside the case carries it to the case, and conduction takes it to the outer surface for outside convection to remove. Heat transfer due to radiation ("infra-red") is negligible (this is why there's not much point in painting heatsinks black, etc. Actually, most paints actually make matters worse, as they're not "black" as far as infra-red is concerned.) Note that since there's not much convection from inside to outside of the case, there is no real advantage to a larger regulator heat sink in terms of overall operating temperature or IC junction (chip) temperature. At best it might help out a borderline regulator and reduce hot-spots slightly. So short of putting big vent holes in the case there's no easy way to improve heat flow from inside to outside of the case.

We can reduce the thermal resistance between the chip and the inside of the case using a metal heatsink to aid in the internal convection process. This is already done for the regulator, which is why it's usually the least of our worries.

To significantly reduce the operating temperature of the ULA and CPU chips, all you need is two strips of aluminum, 1" x 3", 1/16" to 1/8" thick, and a good grade of cyanoacrylate ("super-glue") adhesive. We'll cement the aluminum heat sink to the top of the large chips and solve perhaps 90% of overheating problems.

When you have the materials, open the computer and remove the board (or carefully flip it over so you don't have to remove the keyboard tails.) After thoroughly cleaning the sinks and scoring with emery paper to ensure a good bond, lay each heat sink next to its respective chip, and mark where the chip should be so that the heatsink doesn't stick over

either edge of the board. DON'T try to glue the sinks straight to the installed chips, if any super-glue runs over the edge you'll be in trouble. Instead, gently remove the chips. Lay down the heatsink and apply a thin line of glue between your end marks, and cement the chip to the metal upside down. Let the bond cure overnight, preferably with mild clamping. Be sure you cement the right heatsink to the right chip, as the positions of the ICs on the strip are not the same. Also be careful not to glue them on backwards!

When cured, carefully re-install the chips with attached heat sinks, re-assemble the machine in the case, and test it. If you followed the instructions and used fresh, good-grade cement, the bond will hold even as the chips heat up. If you prefer, you can use a good epoxy instead, just don't make the bond too thick. Silicone cement probably works but I haven't tried it. --A good bet for this would a high conductivity adhesive, RTV silicone or epoxy. TB.

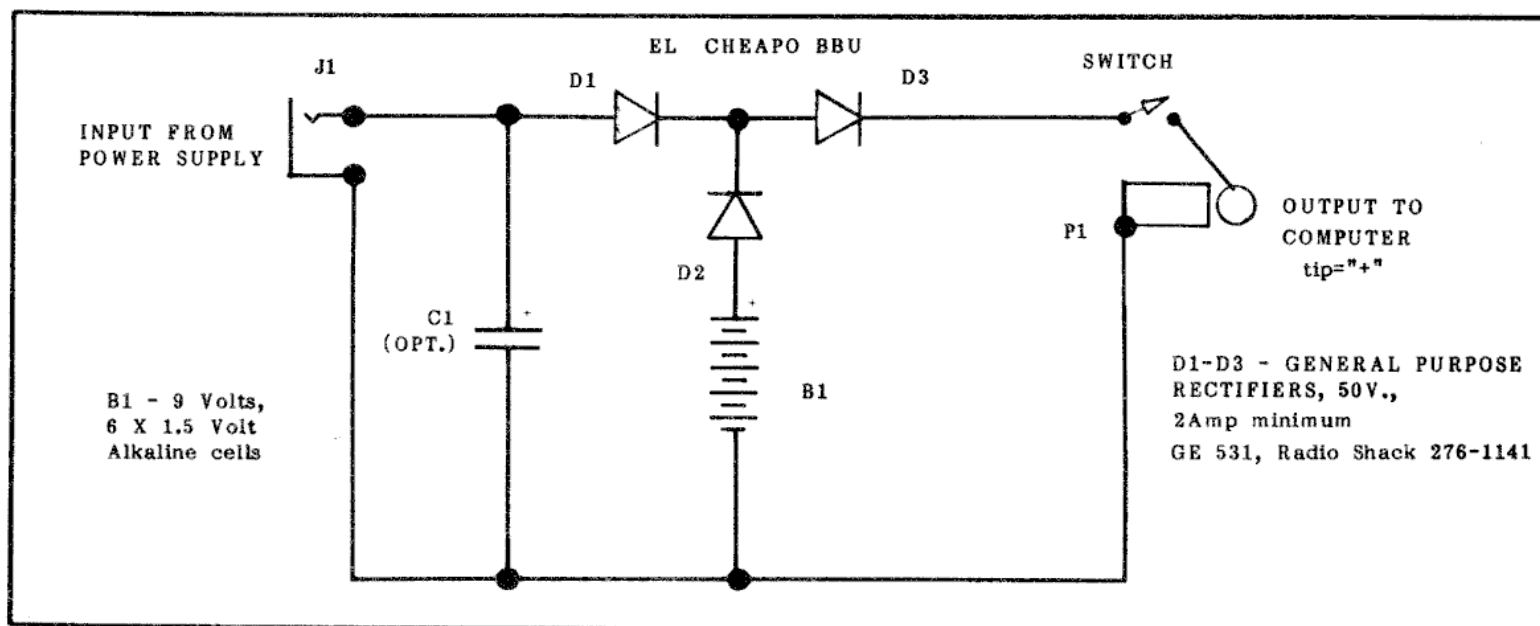
Chances are, you just cured your machine of strange crashes no matter how hot it gets where you live. The following section gives an additional aid if you're still having trouble.

(RE-EDIT NOTES: Since this was written, I've found out that the replacement ULA chips supplied by Timex come with a small ribbed heat sink glued on top, i.e., they come "stock" with "custom coolers" now. It might be a good idea to get one of these as a back-up while they're still available. TS1500 owners - your ULA chip is in the smaller and virtually non-removable "flat-pack" package. The smaller size could aggravate heating problems, and the new package makes replacement a real night-mare. Not only are they soldered in, but the solder joints are UNDER the chip! DEFINITELY consider installing a "custom cooler" on these! this is especially true if you live where the weather gets unbearably hot. Another alternative is to do like I did and move to Canada.)

Battery Back-up The EI Cheapo BBU

The circuit shown is the simplest scheme possible to provide power-down protection for your ZX/TS. You just need a battery holder capable of holding six "one-life" batteries (AA, C or D size,) a few silicon diodes (GE531, Radio Shack 276-1141, or other 50 PIV, 2A silicon rectifier) and an on-off switch. An enclosure, some zip cord and hookup wire, a mini phone plug and jack complete the parts list.

As long as the input voltage from the power supply is greater than the battery voltage (about 9.3 volts or up), diode D1 conducts, and no drain is put on the battery. If the supply voltage drops below the battery voltage, D2 conducts, and the battery keeps the machine going. Alkaline AA cells should run about 40 min., C cells over an hour, and D cells up to 3 hours.



Diode D3 drops the voltage to the computer for decreased regulator heating (and lower overall temperature). You can try two diodes in series for even less heating; most machines run fine down to about 8 volts. If your supply does not provide the minimum 9.3 V. under load, chances are you can bring it into spec by connecting a 1000-2000 uF. 16V. electrolytic capacitor across the input jack J1. Observe polarity! Also, always turn the switch off when connecting/ disconnecting the lead to the computer; even a brief short can damage a diode.

In the next installment I'll show you an enhancement of this circuit by including indicator lights and using rechargeable batteries. After that we'll get into building an entire replacement power supply for more expanded machines.

The Re-chargeable BBU

Now we'll make some changes and additions to "El Cheapo BBU" that will allow us to use rechargeable "NiCad" cells to provide back-up protection and short-term (up to an hour or so) portable use of your ZX81/TS1000 machine. This approach uses your good old T/S wall-plug supply to power the computer WHILE KEEPING THE BATTERIES RECHARGED. This works fine for most machines, at a very reasonable cost. Both my machines have such circuits built-in "under the hood." My 16K workhorse (my original ZX81) supports MEMOTEXT and CIF as well as the RAM pack, and has been "power-line proof" for well over a year. The other machine supports a 64K RAM and video driver/ reverse board. Both still have about 75 mA. leeway before overtaxing the wall supply. Both are on all the time though they have an ON-OFF switch (for crash recovery). These machines are the "prototypes" for this and future installments of THE CUSTOM T/S, so I'll be referring to them now and again.

For larger systems (more stuff hanging off the edge connector) you may need to build a larger supply. This is a snap. Virtually every home micro magazine has published power supply circuits. But while you're at it, why not include battery backup protection and allow portable operation with your supply? I'm amazed that there has been so little coverage of this, since it is a very simple addition. [Since this issue came out, almost EVERY

mag has run such an article - I'm not saying we were copied, but we were one of the first to come out with such a circuit - ed.] The technique is by no means limited to the T/S machines, but the discussion here assumes the ZX81/TS1000/TS1500. [See SWN Vol. 2 for modifications for TS2068 -ed.]

So here we go. Finally you'll be able to thumb your nose at those times when your trusty power line poops out.

RE-CHARGEABLE BBU

Here's an expansion of "El Cheapo BBU" which uses readily available GE or Radio Shack rechargeable NiCd "AA" cells (rated 1.2-1.3 V. at 500 mA-Hr) to provide a battery backup which can also power your computer portably at minimum cost. In this case, however, the circuit needs at least 9.8 volts at the input to insure that the battery will remain at full charge during normal operation. With the power supply provided with the ZX81 or TS1000, this corresponds to a current demand of up to about 550 mA., or the basic machine plus RAM pack and perhaps one or two low-drain accessory boards. Before you go ahead and buy parts for this circuit, it would be wise to check the power supply voltage with your computer running and loaded with the accessories you're using; if you get less than 9.8 volts you should consider the UPS option that follows.

Assuming that you get about 10.2 volts, as I do on my machines, this circuit will give you reliable and compact protection, running your computer for about an hour with the AC off and batteries fully charged. This is, in fact, the circuit for my former "BBU-1," and I might add that as far as I know, none of these have ceased to function. As another side note, if you bought one of our BBUs and it HAS failed, do let me know; I will of course continue to provide support and service even though we're no longer building them.

The circuit is almost identical to "El Cheapo;" the significant addition is resistor R1, which provides the charging current for the battery (7 NiCd "AA" cells in series). The value shown, limits the current to a safe level, and will fully recharge the battery (with computer off) in 14-16 hrs.

R4, zener diode Z1, and LED2 are an indicator circuit that shows power delivery to the computer. As battery voltage drops, this LED will grow dim, extinguishing near the point where the machine crashes (about 7 volts). Actually this circuit is slightly redundant, since you can tell when the machine is about to crash when the TV picture starts to wash out (lose contrast). So you may omit these if you wish.

The remaining parts (Q1, R2, R3, LED1) are an indicator circuit that shows when voltage is being delivered by the supply. More accurately, it is a "charging" indicator - LED1 only lights when current is flowing through R1 (and the battery is charging). We have to use this slightly more complex circuit as we can't afford another diode at the input (for voltage reasons) to prevent the battery from lighting LED1 through R1.

Just about any PNP transistor will do for Q1. If you use a silicon transistor, LED1 will go out when battery charge current drops below about 10 mA. With germanium units the LED will be lit down to about 4 mA.

As with "El Cheapo BBU" BE SURE to switch OFF before connecting the output plug to the computer, as a momentary short could blow D2.

Before using your unit after construction, you should first "cycle" the battery 3-4 times to break it in (when new, NiCd cells frequently show abnormally high terminal voltage). Alternately charge the battery (computer OFF) for 24 hours, then discharge it by powering the computer with AC off until LED2 grows dim or the computer crashes. Then

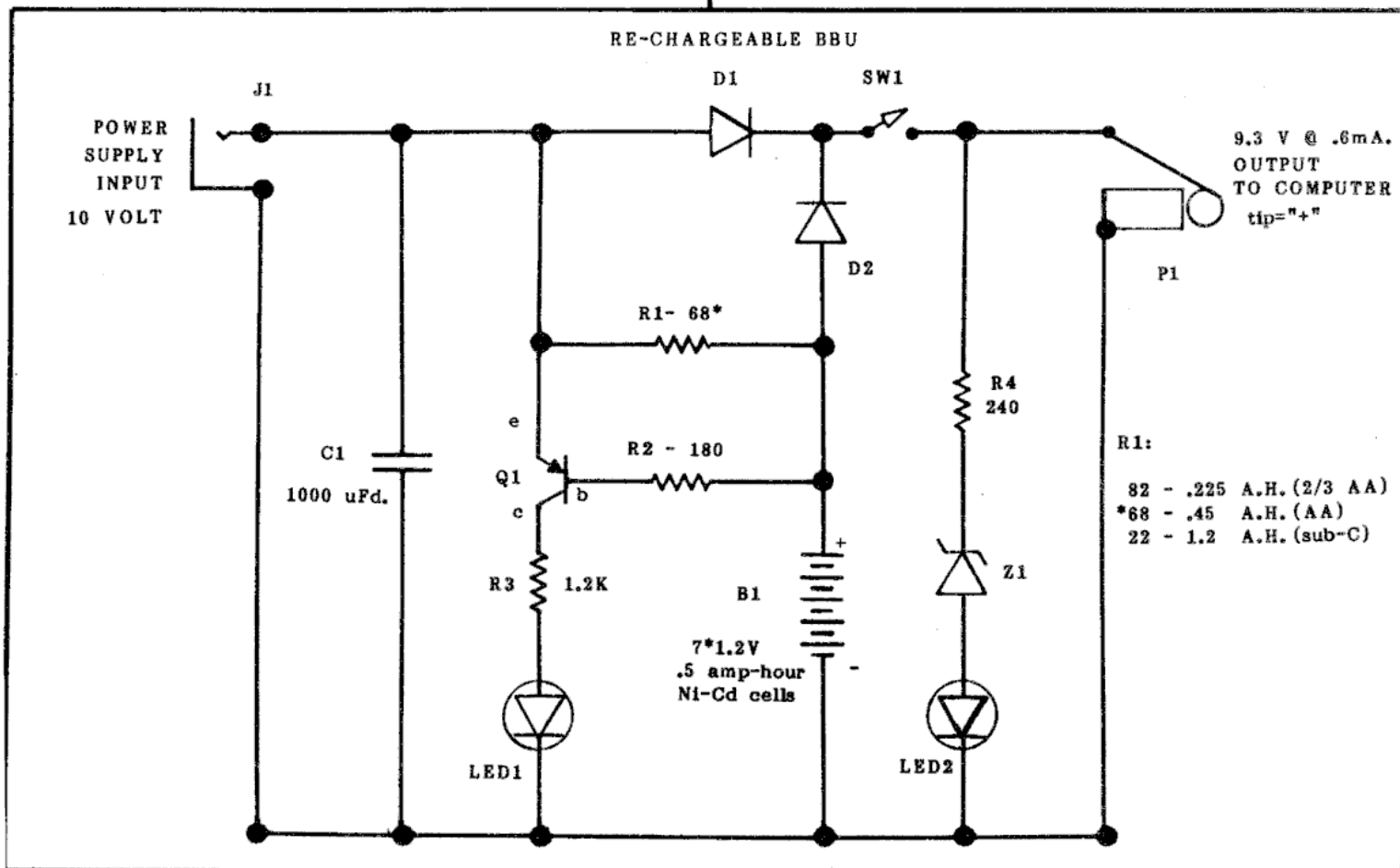
give it a final 24 hour charge, and you're all set - you can now leave the computer on at all times. (Same goes for built-in BBU and UPS described below.)

Parts List

B1 - 7 Nickel-Cadmium "AA" cells (.5 Ah) wired in series
 C1 - 1000 uF., 16 V. electrolytic capacitor
 D1, D2 - General purpose silicon rectifiers, 50 PIV, 2A (GE 531, R/S 276-1141 or eq.)
 J1 - 1/8" phone jack
 P1 - 1/8" phone plug
 LED1 - Red LED, any size/shape
 LED 2 - Green LED, any size/shape
 Q1 - General purpose PNP transistor, 2N4403, 2N3906, etc.
 R1 - 68 ohm, 1/2 W resistor (see note *)
 R2 - 180 ohm, 1/4 W Resistor
 R3 - 1200 ohm, 1/4 W.
 R4 - 240 ohm, 1/4 W.
 SW1 - SPST slide or toggle switch, 1A rating or greater
 Z1 - 1N5228 or 1N5229 or eq. (3.9-4.3 V, .4 W. zener)
 Battery holder (if cells do not have solder tabs)
 Enclosure
 Zip cord (for output cable), wire
 Perf board

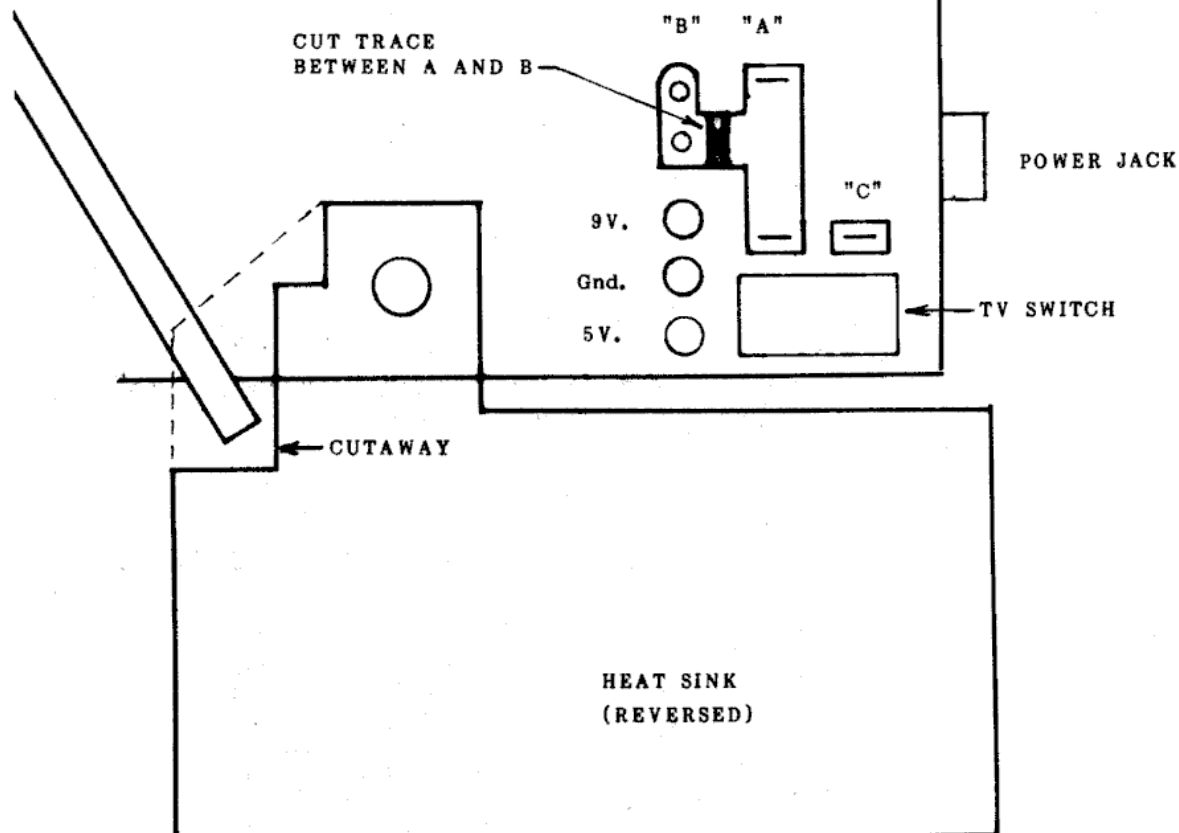
* R1 Note:

82 ohms - for .225 Ah. "2/3 AA" cells
 68 ohms - for .5 Ah. "AA" cells
 22 ohms - for 1.2 Ah. "sub-C" cells
 Consumer "C" and "D" cells (GE, Radio Shack) cells are "sub-C" (1.2 Ah.) in a larger case.



Built-In BBU

PC BOARD



Built In BBU

Yes, folks, there's plenty of room left in even your tiny "stock" case to include re-chargeable BBU INSIDE THE COMPUTER. This makes it immune even to pulling the plug! My 16K machine has the BBU circuit, exactly as above, "under the hood." I must say that using AA cells makes things rather tight, and it required some case re-fitting. I got a little wiser when I customized my 64K TS1000; by using what's commonly called "2/3-AA" cells (rated 225 mA-hr) and everything fits fine. Backup protection is only up to about 25 minutes, but this is plenty of time to find a candle and SAVE to tape if the lights go out.

Finding these mini-AA cells could be a bit of a problem, as they're primarily an industrial component. We still have a few sets of these left (they're the cells we used in Baby BBU) if some of you are interested, and can get more if many of you want them. Cost is \$16.00 ppd. per set (7 cells), wired in a pack and ready to go.

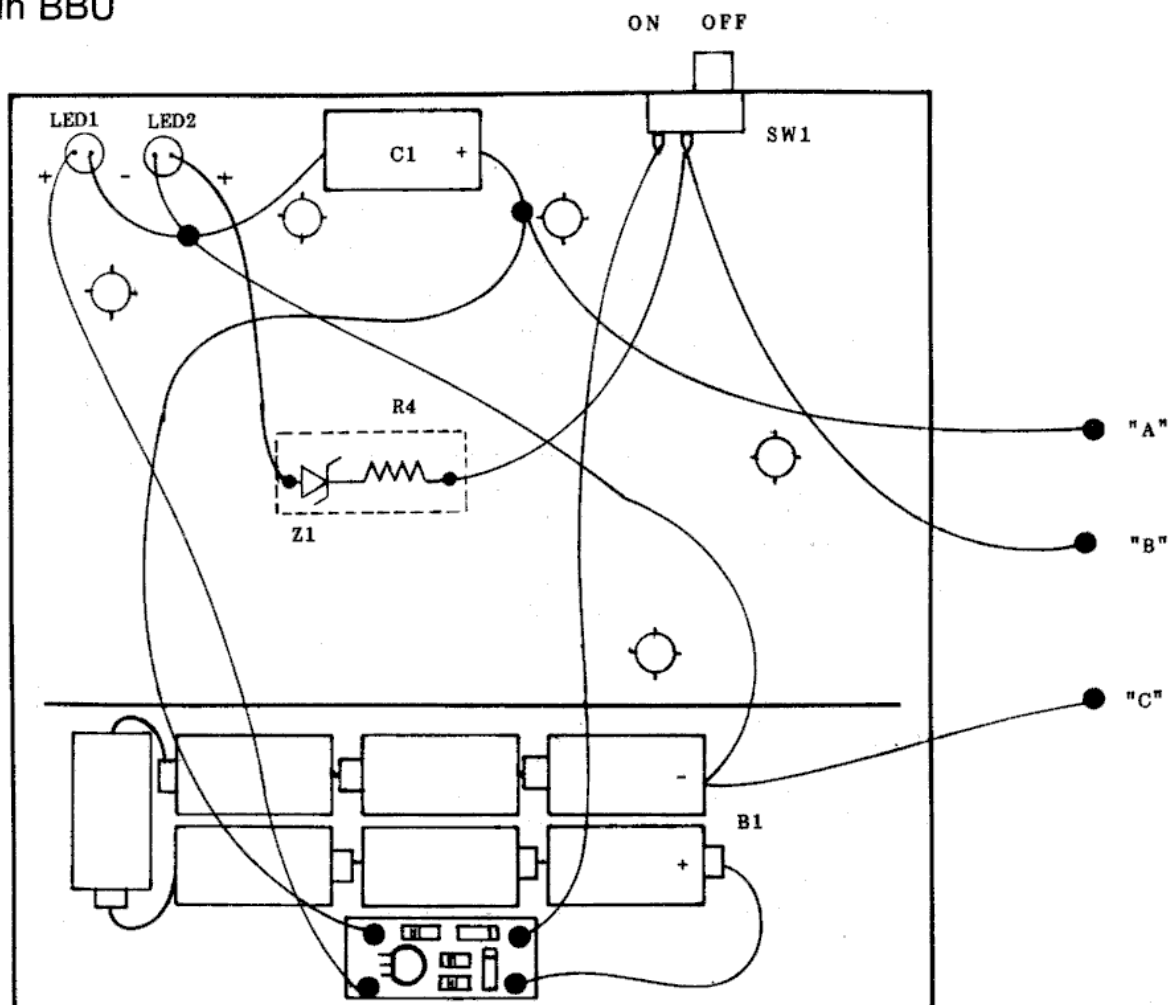
But where to put this stuff? Remove the bottom case-half from your machine. You'll see a big open space under the keyboard - plenty of room if it weren't for the regulator heat-sink. So what we'll do is turn the heat sink around so it faces the other way and leaves us PLENTY of room. Unbolt the sink, and flatten it out with a hammer (it has a slight bend which is really un-necessary). We'll then re-install it "upside-down", but first you'll have to cut out a small notch near the regulator

hole (with a hacksaw or "nibbler," finish with a file) so that the reversed sink will clear the grounding strap (and won't short any circuit traces on the "Issue 2" -ZX81 board).

Now is a good time to make the subassemblies; the perf board and the battery pack. Build up the circuit including the resistors, transistor, and diodes on a 1"x1" piece of perf-board. You can omit R4 and Z1 from the PC board, and just solder them in series with flying leads off the ends, and the assembly covered with "shrink tubing" or tape; connect this slightly lumpy wire between the switch and LED2 (if you're using LED2 at all). Provide flying leads to connect to the other stuff (+ to battery, circuit input and output, LED1). Mounting the battery and board is easy with "sticky-foam." Mount the LEDs in whatever holders you like in the upper right corner of the top case half (above the edge-connector cutout). Where you put the switch depends on your tastes and switch size, I suggest a miniature slide switch mounted on the rear apron of the upper case-half. This might present difficulty with some plug-ons, but even with my Memotech stuff there is enough clearance to reach in with a metal ruler to flip the switch. (You don't want to make it TOO accessible!) You can file the switch post down if need be, or you may modify the channel selector switch to act as "power on-off."

Capacitor C1 should be 1000 uFd. or greater at 16V. It should be no larger than about 5/8" diameter. It can be super-glued to the case either next to the battery (if it's not too fat) or at the rear of the case between the center post and the modulator.

Built-In BBU



Finally, you'll have to cut the main 9V trace on the computer board between the power jack and the rest of the circuit (see diagram) to allow you to put the BBU circuit in-line. A total of 3 wires go to the board at the power jack; "from supply," "to computer," (the other side of the cut trace) & "ground."

Now, it's just a matter of hooking everything up. Check and double-check your connections. ARE ALL POLARITIES CORRECT?! The diagram should help, run your lines as shown but dress them along the corners for a neat appearance. re-install the board, connect the three wires to the computer power jack, and plug in the power supply for the "smoke test." LED1 should come on, and LED2 should glow when you flip the ON switch. Computer with built-in back-up is now complete. Give it a final test, if you were careful and used good parts you now have a machine with features not found on anything within 10 times the price. (By some strange synchronicity we are having our first electrical storm of the season as I am writing this. I ain't worried! And now you won't be, either.)

Un-interruptable Power Supply

Onward to replacing our good old wall-plug power supply with something more substantial. You have several ways to go, and I'll try to touch on some of them here. BE CAREFUL as in this project we'll be dealing with "raw electricity." Not to scare you off, but "follow the rules" about insulating connections, observing polarities, and just generally not being careless. Don't blame me if your "smoke test" actually fills the room with nasty PVC smoke. The stuff we're doing here is really so simple that I can't foresee any major problems, but if you are having trouble, do write or call and I'll do what I can to get you back on the rails.

Perhaps the simplest approach is to get a 12V car battery and a trickle-charger to go with it (1 or 2 amp. jobbies, I recently got a 2 amp. "Mity-Mite" from a discount auto parts for \$14.97). Oh, yeah, you say. Then put a string of diodes in series

to drop the voltage to the computer. While this works, it is DEFINITELY NOT (!!) RECOMMENDED. The first time you short the output you'll have a string of "fry-odes" (short circuits) instead, and the system sees over 13 volts, causing regulator overheating and possible RAM failure; goodbye TeSi. The alternative is almost as simple, and costs about a buck more than 6 good diodes. This is a miracle device called the fixed 3-terminal regulator. They are available in a "7800" series (7805 - 5 volts, 7808 - 8 volts, etc.) which provide about an ampere of regulated output, and in an "LM340T" series (LM340T5 - 5V, etc.) which provide up to about 1.5 Amp. Both are in the popular "power tab" package, but LM340K is a TO-3 equivalent (diamond-shaped two-hole package) also available. An similar adjustable unit is the LM317, another is the LM350 which allows up to 3 amps. These are all accurate and dependable, best of all they're protected against excessive current AND temperature, so are output short proof. Virtually indestructible. Such a device (7805) provides the regulated 5V inside your machine. As a side-note, many borderline machines benefit by substituting an LM340T5 for the 7805. If we pre-regulate the 9V in, we'll have "double protection" against crash-producing glitches.

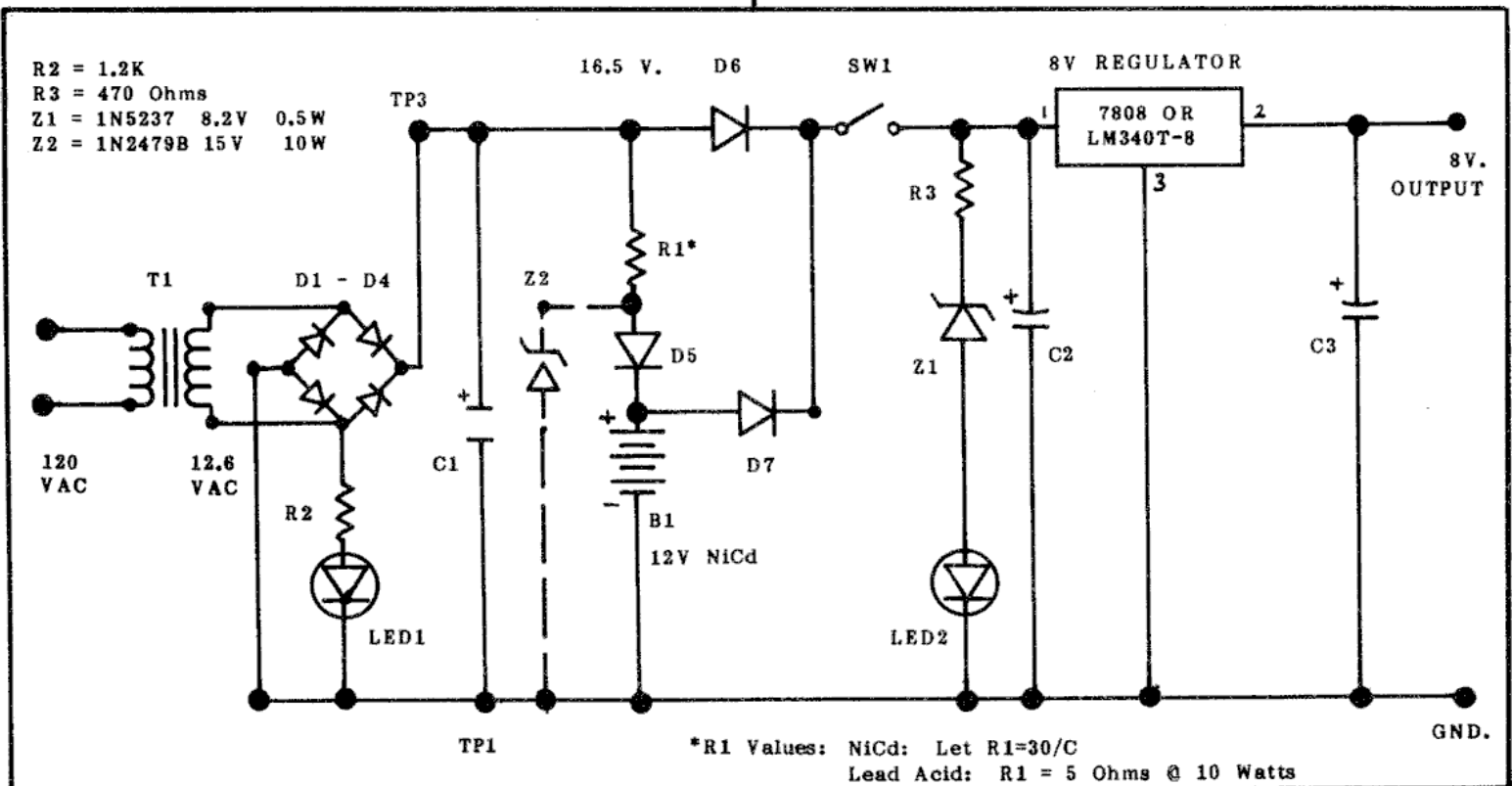
How these are used will be clear by the time we're through, but first let's consider some other alternatives. You could replace the car battery with a motorcycle battery, or better yet a 12V "gell-cell" of 4 - 12 amp-hr rating. Add a charge limit resistor, filter capacitor, a couple diodes and indicators, and you have a UPS. The circuit shown is a general one, usable with transformers of 12 VAC 1-3 amp (or 25.2 VAC center-tapped as in most car battery chargers. Takes two less diodes. See diagram.)

Another side-note about car-battery chargers; they usually have a circuit breaker to prevent burn-out due to shorts, and already come with two hefty diodes (4 in untapped 12.6 VAC units) and a case. There's even room for ten "sub-C" Nickel Cadmium's for protection up to an hour (at 1 amp. - up to 2 hours for basic machines + RAM.)

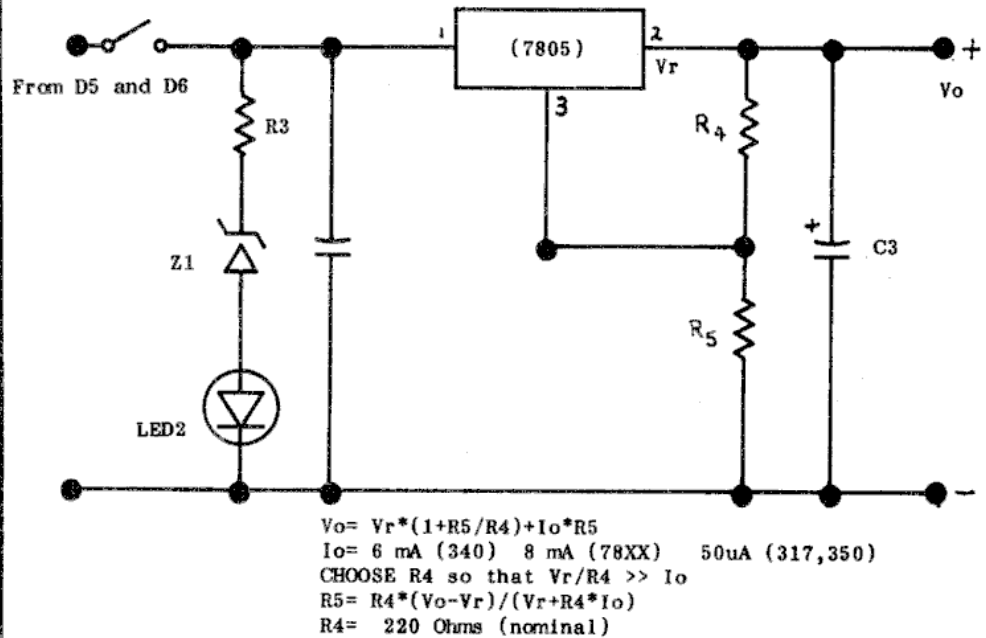
Or you may prefer to start from your choice of transformer. In choosing a transformer, you trade off between weight/size/cost and power. If you're powering a machine + RAM only, 12.6 VAC 1A is more than enough. To be on the safe side choose 12.6 VAC 2A. If you're REALLY power hungry get a 4A unit and use it with an LM317K or LM350. You will also have to choose diodes and regulator to handle the current.

But a graphic is worth 0400h words, so let's look over the schematic and other scratchings. There are four main elements; transformer/ rectifier, filter (C1), battery charge/switching (R1, D5-7) and regulator. The first section might be a car battery charger, the filter capacitor should be 2000 uFd. or so for .8 A. output, 4700 uFd. for 1.2 A or more. The idea is to get about 17 (no less than 16) volts raw DC at TP1 under full load, increasing C1 might bring it up to spec, if not you're overtaxing the transformer or have a baddie.

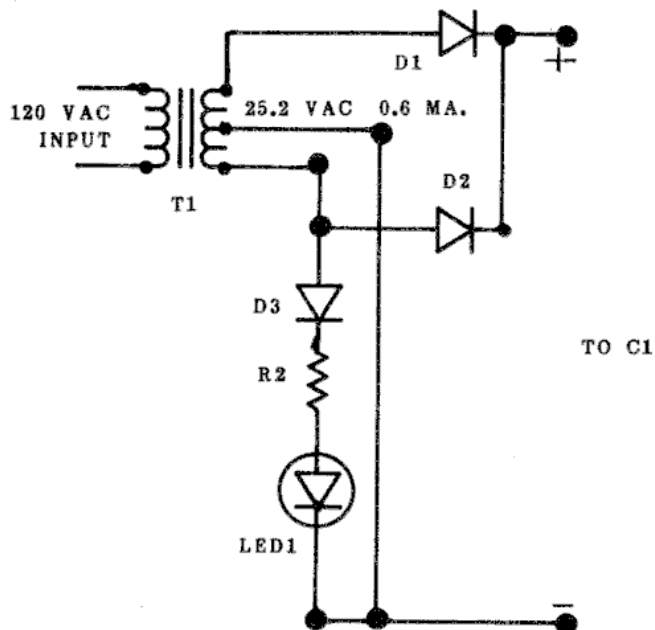
The charging circuit you've already seen. The D5/R1 combo is for NiCd cells; for small sealed lead acid (gel-cell) add a 15V, 5W or up zener diode between TP2 and ground (battery minus). This prevents overcharge damage with this type of battery.



A 5V. Regulator can be used to provide 8 to 10 VDC, as shown below. R4 and R5 are not needed with the 8V regulators.



Alternately, you can use a 25.2 Volt, 0.6 Amp. transformer with the connection shown below. The transformer must have a center tap.



That leaves the regulator, which is the easiest part. If you're running from a car battery, this and the switch is all you need; use the charger as-is to keep the battery at full charge (battery itself provides ripple filtration, no need for C1). Schematic tells all. Small capacitors are recommended at input & output, .1 - 1 uFd. disc ceramic or tantalum. Note the alternate circuit - most machines work fine (and with less internal heating) at 8V, but some RAMs are touchy about this so you may want to trim it up to 9.5 volts or so. Experiment with R5 to get the voltage you want.

May the Power be with you!

An Improved Joystick

By Cedric Bastiaans

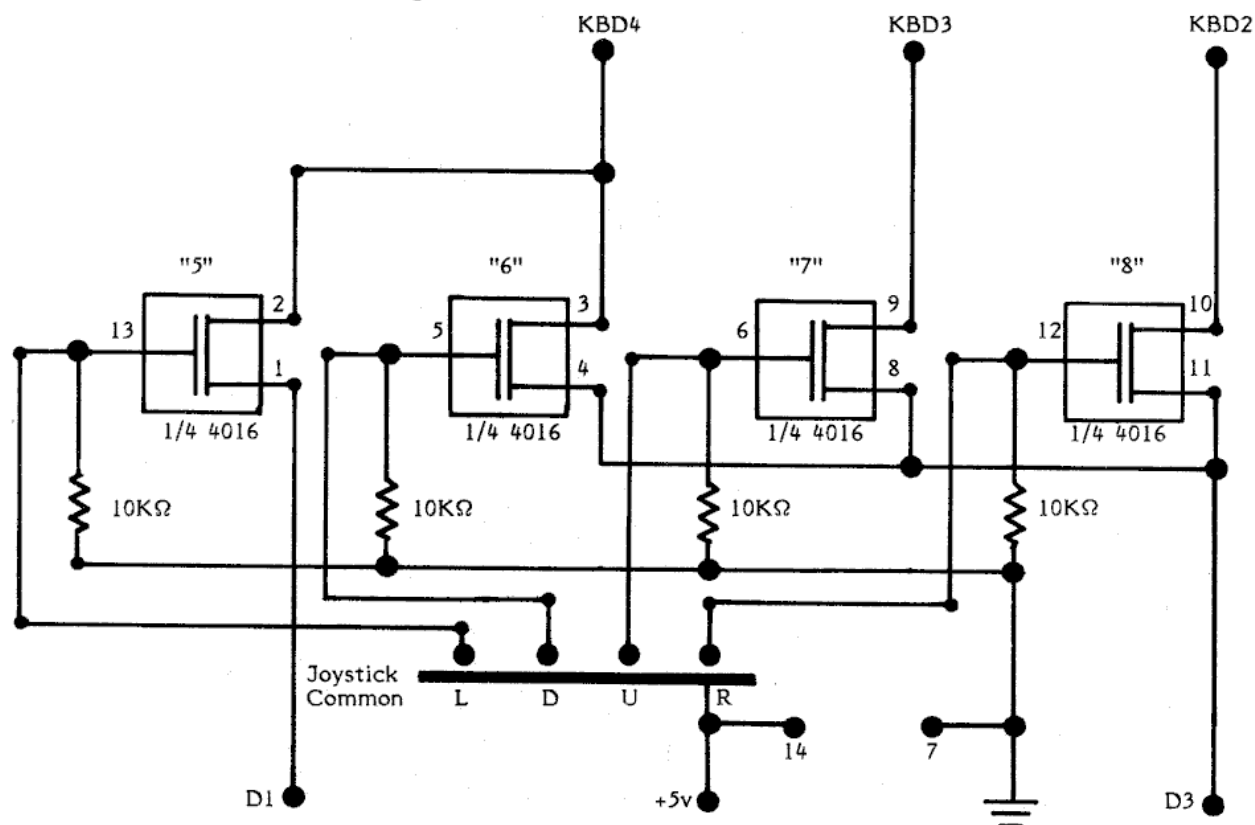
It was interesting to see in page 10 of the 1:1a issue of SyncWare News, a joystick circuit that is a variation of a circuit which I have been using since I customized my Sinclair ZX81 in the summer of 1982. When I moved the ZX PCB in a large keyboard case, together with a HiTek 53-key professional keyboard, I brought the 13 KBD and DIODE lines out on a 16-pin connector, together with +5V and ground.

I have since been using encapsulated circuits as per the schematic below. An Atari joystick plugs into the circuit, which in turn plugs into the 16-pin connector. The circuit uses a Quad Analog Switch, such as the 4016 or 4066. If you've experimented with this, you know that the KBD/Diode matrix does NOT require a low-resistance connection for a character to be formed. As a matter of fact, the "contact" resistance can be as high as 1500 ohms for reliable operation. The "ON"-resistance of the logic switches is anywhere from 100 ohms to several hundred ohms, depending on the "enable" voltage. The circuit has the advantage over the transistor version that no experimentation with resistor values is required, and it is cheaper; a quad switch typically sells for 99 cents, against four transistors for at least two dollars... [Of course, that depends on how and where you buy your parts - ed.]

So far, I have not had a need for more than four control lines per circuit, but you could of course use more devices to give additional control lines.

EDITOR'S NOTE - An additional advantage to this type of circuit is that it has somewhat better noise immunity than the transistor circuit. If you have to remotely control the keyboard from a great distance, then the recommended technique is to use opto-couplers to do the switching, and the lines that carry the DC to the LED portion of the coupler can be practically as long as you like. The coupler effectively isolates out any ground-loop-induced noise, etc.

Improved Joystick
Using the 4016 or 4066 Quad Analog Switch



Intro to I/O

Here are a couple articles that introduce you to control applications for the T/S computers (ZX81, TS1000, TS1500) with a minimum of effort and (best of all) at reasonable cost.

Before we control something with a computer, we usually need some way of getting information back to the computer; i.e. to control an oven, the computer has to know the present temperature. Getting analog information (smoothly continuous like in the "real world") into digital form (on or off, nothing inbetween) is called, aptly enough, "analog to digital conversion" or more simply A-D (read "A to D.") Several approaches are possible, but as usual there are trade-offs. The faster and more accurate you want the data, the more hardware it will take (and the more expensive it gets). A very sophisticated data acquisition system is available from Eric Reiter's Computer Continuum (301 16th Ave., San Francisco, CA 94118) for about \$200; software is also available to literally turn your ZX/TS into a multi-function storage oscilloscope, chart recorder, spectrum analyzer etc. The board can be adapted to other machines as TS2068, Apple, etc. This is the way to go if you need that kind of performance; and the price is considerably less than similar add-ons for other machines.

But wait! There is another way to go. For \$42.95 incl. shipping you can get a VOTEM kit (Down East Computers, PO Box 3096, Greenville, NC 27834) and get accurate single-channel data acquisition. (Just a little later, though, we'll show you how to get several channels.) VOTEM uses a conceptually simple approach; convert the analog voltage to a frequency, then use the computer to count how many times the signal changes state during a loop of preset length. So all it really takes is a V-F converter of some sort and a short machine code routine to read the EAR jack and keep track of the counts. This simplicity makes this approach easy to use, plus you can trade speed and accuracy as required for your application.

VOTEM is more than this, though; since it's easy to understand and comes with extras like a tape-signal conditioner, temperature probe and a friendly, very complete text, it is a powerful educational tool. Only high quality components are used, the V-F converter itself is very accurate and stable. You can get the manual separately if you'd rather build your own from the ground up, but it's hard to beat Down East's price, all things considered.

So now you have a way of getting an analog voltage into your computer memory. From there you can print it, plot it, manipulate it any way you want. But how about going the other way? How can you get your computer to control other devices in response to software command (such as turning on a heater when temperature drops below a certain point?) Again, there is a relatively cheap but effective add-on available; the BB-1 control module from Byte Back Co. (generally referred to as a "Byte Back Module" although the company has other products, including memory packs with attractive specs and price tags). This allows you to control eight devices with a simple POKE. It includes on-

board relays for isolation and (relatively) high power switching. Tom and I both have one and find it an excellent product; more about this module elsewhere in this issue.

Meanwhile, there's another option available to you. After receiving a letter from Chris Bannister, in which he described his "Blue Box" (built by Paul Hunter) which places the four 2K blocks of Hunter board memory under software control, I wrote to Dr. Hunter to ask if he'd be interested in sharing the details. He promptly responded with the beautifully written article you will find in this issue. If you have a Hunter board (if you don't, you should), then you can use the output of the circuit to select 2K blocks and duplicate the infamous Blue Box, or you can control whatever your heart and application desires. How about selecting one of four inputs to your VOTEM? (Aha! I can hear those wheels turning now.)

The final step in interfacing your computer to the outside world is providing the necessary isolation and power amplification required to operate "real world" devices as motors, heaters, lights, etc. Most of you are familiar with relays, which use an electromagnet to close a mechanical switch. The switch contacts are electrically isolated from the coil, and handle much greater current than is required by the coil; it is thus both an isolator and amplifier. But even the relay coil takes far more current than can be provided by the usual digital IC's; additional amplification is required in the form of "relay driver" circuits. An improvement is the "solid-state relay." The heart of such devices is the "opto-coupler." An opto-coupler consists of an LED and a light-sensitive transistor in the same package but not electrically connected. The transistor controls, in turn, a larger transistor, which is the actual load switch. This approach is much cheaper than mechanical relays, and in many ways more reliable. For moderate loads (up to a few amps) each SSR costs only about \$5 max.

As a prelude to Paul Hunter's article, I present a modification to VOTEM which adds yet another feature to this potent little "out-board"

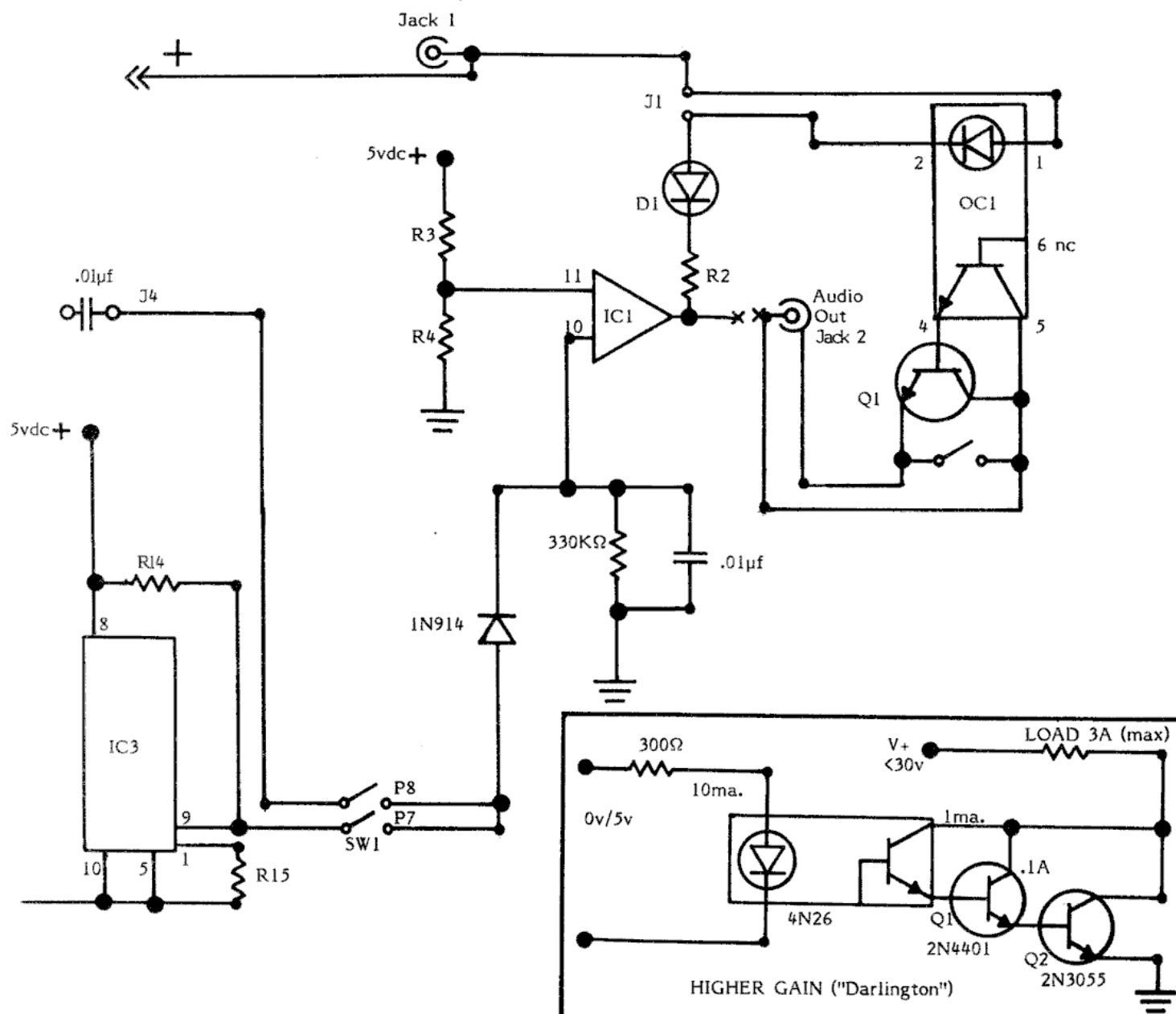
Votem Cassette Controller

If you have a VOTEM, or if this article talks you into buying one, here is a simple modification which will extend the board's utility further, and allow you to start LOADING a program from tape, flip a switch when the loading bars appear on the screen, and walk away; when the program has loaded, the tape recorder motor is automatically shut off, and the tape remains there until you come back and flip the switch back on. If you don't have a VOTEM, you can still use the information given about opto-couplers in your own interfacing projects, or add it to one of the circuits shown in "Cassette Connection." But if you already have a VOTEM, you can get right on with it, or incorporate it into your unit as you're building it up.

- 1 - mini-plug (goes to VOTEM)
 - 1 - sub-mini plug (goes to cassette REM jack)
 - 1 - optocoupler (high gain phototransistor type)
(4N35, 4N36, 4N37, H11A1, TIL117, or eq.)
 - 1 - medium power NPN transistor, e.g. R/S 276-2030)
 - 1 - SPST miniature slide switch
 - 1 - .01 uF. disc capacitor
 - 1 - 330 K-ohm resistor, 1/8 or 1/4 watt.
 - 1 - small silicon diode, 1N914 or similar
- Cable, hookup wire, epoxy or superglue.

Radio Shack sells an assortment of five couplers for \$1.98 (276-1654) which will give you several to experiment with. There are basically two types of couplers; transistor (DC only) and triac (designed for switching AC) types. Transistor units are also available in a "Darlington" configuration (two transistors in cascade for higher gain) which require less LED drive current (higher overall gain). For our application, though, the ordinary transistor type is better as it has a lower on-state voltage drop than Darlington's. The transistor Q1 should be capable of handling up to about 500 mA. and have a minimum gain of 50 or better. Our LED drive current will be about 7 mA., so with a coupler gain of .5 and transistor gain of 50 (worst case) we get a maximum output current of about 175 mA. Most recorders take under 100 mA. in "play" with the speaker disengaged, so we should be safe. If your circuit works but the motor runs slow or wows, try a

Figure 1. VOTEM Modification for auto-tape shutoff



higher gain transistor and/or coupler. (Couplers such as TIL111, MCT-2, 4N26-28 typically have a lower gain, .1-.2, so will require a higher gain transistor).

Here are the step-by-step modification instructions:

1. Remove capacitor C1 and jumper J4. Install C1 (.01 uF.) where J4 was.
2. Remove D2 (not really necessary and reduces accuracy. You're better off without it. Down East Computers also recommends its removal to improve accuracy.)
3. Cut J1 in the center and bend the ends up. This will be our connection to the opto-coupler LED.
4. Cut the ground trace on both sides of JACK 2. If you wish, you can bridge the gap with a piece of insulated wire, but it's not really necessary.
5. Cut the trace going to pin 10 of IC1, before the juncture to pin 8 and the DIP switch. (see diagram). Bridge the gap with a 1N914 or like silicon diode, banded end (cathode) end toward IC1 pin 10. Connect a .01 uF. capacitor and the 330K resistor from the diode cathode/ pin 10 to the adjacent ground trace.
6. Connect the emitter (pin 4) of the coupler to the base of the transistor, and the collector of the coupler (pin 5) to transistor collector. Connect the LED portion of the coupler (pins 1 and 2) to the ends of J1, with pin 1 (anode) going to the one closer to the edge of the board. Don't connect anything to pins 3 and 6.
7. Cement the slide switch to the board between JACK 3 and JACK 5, with the handle protruding slightly so you can reach it with the covers on. Connect it directly across JACK 2. Be sure that the case of JACK 2 is indeed disconnected from VOTEM ground.
8. Connect the free end of Q1 (emitter) to JACK 2 negative (case) and the collector and OC pin 5 to JACK 2 positive (tip).
9. Make up the remote cable. The REM jack on the recorder is usually a sub-mini phone jack, so you'll need a cable that goes from a sub-mini plug on one end, and a mini plug on the other. Make up the cable rather than buying a molded one, as you have to check polarity. Remove the outer cover from the sub-mini plug and insert into the REM jack on the recorder, place the recorder in PLAY. It shouldn't run as the plug is inserted, but you should measure about 6V (or whatever battery voltage) across the exposed connections of the plug. Note the polarity: if the "tip" is positive, connect the cable as usual = tip to tip, case to case. But if the tip is negative with respect to the case on your machine, reverse the connections. The point is that the "tip" of the mini-plug going to VOTEM must be positive.

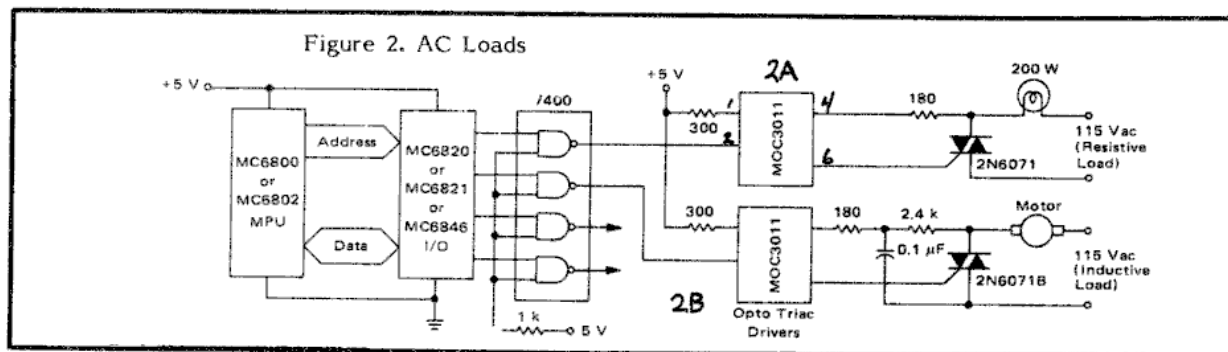
TESTING THE MODIFIED BOARD

Hook up your VOTEM as usual for tape loading. Don't hook up the remote cable just yet. Verify that the tape conditioner works by playing a program tape and entering LOAD "x". The LED should glow steadily when program data is playing, and go out during silent parts or when tape recorder is stopped. Adjust volume so that the LED only flickers slightly during the pre- and post-save buzz. You can now tell exactly what's going on with your tape signal.

If all is well so far, connect the remote cable. Flip the slide switch ON, and the recorder should work as if nothing had changed. LOAD an actual program tape, and when the actual program starts (LED on) flip the switch to "auto-off." The motor should keep running. Motor current is now passing through the switch transistor, which is energized whenever LED is on by the opto-coupler. As long as the data keeps coming, the motor stays on. At the brief silence at the end of the program, the LED goes off and so does the motor. Also, if there is a drop-out, the motor will stop so you can inspect the tape at that spot. The time constant of the R-C filter we added is fast enough to catch even short lapses, while providing enough filtration to insure the LED driver comparator stays on during signal.

So you see how easy it is to use opto-couplers, and you have another handy feature on your computer system. Opto-coupler/transistor combinations like this can be used to switch any DC load. If you need more gain (output current) you can use Darlington couplers or extend the external darlington-connected transistors. The more transistors you cascade, though, the greater the on-state voltage drop will be; figure about .4 volts * (N+1) drop, where N is the total no. of transistors (including the one(s) in the coupler).

Coupling to AC circuits is just as easy, and triacs can be cascaded as far as you need with no change in on-state voltage, so you can control as big a load as you like. Example, a MOC3011 coupler can trigger a sensitive-gate 4 amp. triac like 2N6073 directly, or the 2N6073 can trigger a 10 amp MAC11-6 (Motorola part numbers). Circuit 2a shows the simplest approach, workable with resistive loads like lamps and heaters. For motors and other inductive loads (even equipment with large power transformers) you'll need to use something like circuit 2b, which includes a "snubber" network to prevent the inductive "kick" from the load from falsely retriggering the triac. Exact values depend on the load; start with those shown and increase C1 as necessary if the triac gets stuck in the ON state.



Paging Control

by Paul Hunter
1630 Forest Hills Drive
Okemos, MI 48864

One of the advantages of building a ZX81 from a kit was that you got a schematic with the manual, and you gained some insight into how the machine worked as you put it together.

It was evident, for example, that the computer was missing some memory from 8K to 16K -- an area transparent to the BASIC operating system. An expansion to a 16K ROM was probably intended at some point. In any event, the gap has been filled, and overfilled, by the many manufacturers of peripherals for the computer.

This article, possibly one of a series if the response is favorable, describes a very simple way to page or select from several devices using the 8K-16K region. This method is not sophisticated and serves as an easy introduction to I/O control.

The apparent simplicity of the ZX81 design conceals a complicated operating system. Some features of this system make I/O less useful than it could be in any other "normal" Z80 system. For example, the use of NMI (nonmaskable interrupt" in the display routine precludes its use in I/O control. Similarly, the use of I/O addresses by the system is extravagant. Fortunately there are some addresses left and if necessary indirect addressing is possible. (This means that, although the lower half of the address bus is normally used to decode one of 256 I/O ports, the higher half of the address bus can also be used.) For example, OUT C,r places the contents of [register] B on A8-A15, the contents of C on A0-A7, and the contents of register r on the data bus D0-D7.

The addresses used by the system can be found by searching the ROM for IN and OUT instructions. I/O addresses are used in keyboard servicing, TV display routines, printer input and output latching, and in cassette LOADING and SAVEing.

Addresses monopolized by the system are:

- Any address with A0 low, e.g. FE
- Any address with A1 low, e.g. FD
- Any address with A2 low, e.g. FB and the address FF

The final hindrance to I/O operation is the absence in the ZX/TS1000 BASIC of any IN or OUT command. Machine language subroutines must be used.

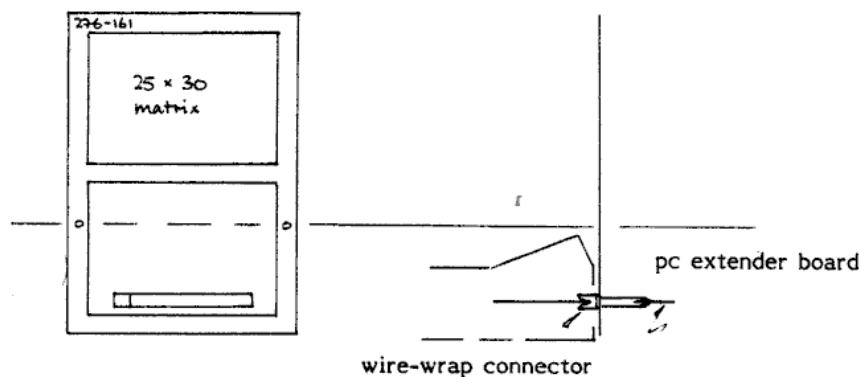
You may well wonder then, why not memory map the I/O and avoid all these problems. Then, for example, the BASIC commands PEEK and POKE can be used to service the port. Well of course you can -- particularly if you haven't already got a full complement of memory (64K). The same kind of circuit can be constructed.

HARDWARE PROJECTS

Building hardware projects can become addictive -- it is very satisfying to see something you build actually work, and in many respects the ZX/TS makes it easy -- most of the Z80 CPU signals are available at the edge connector.

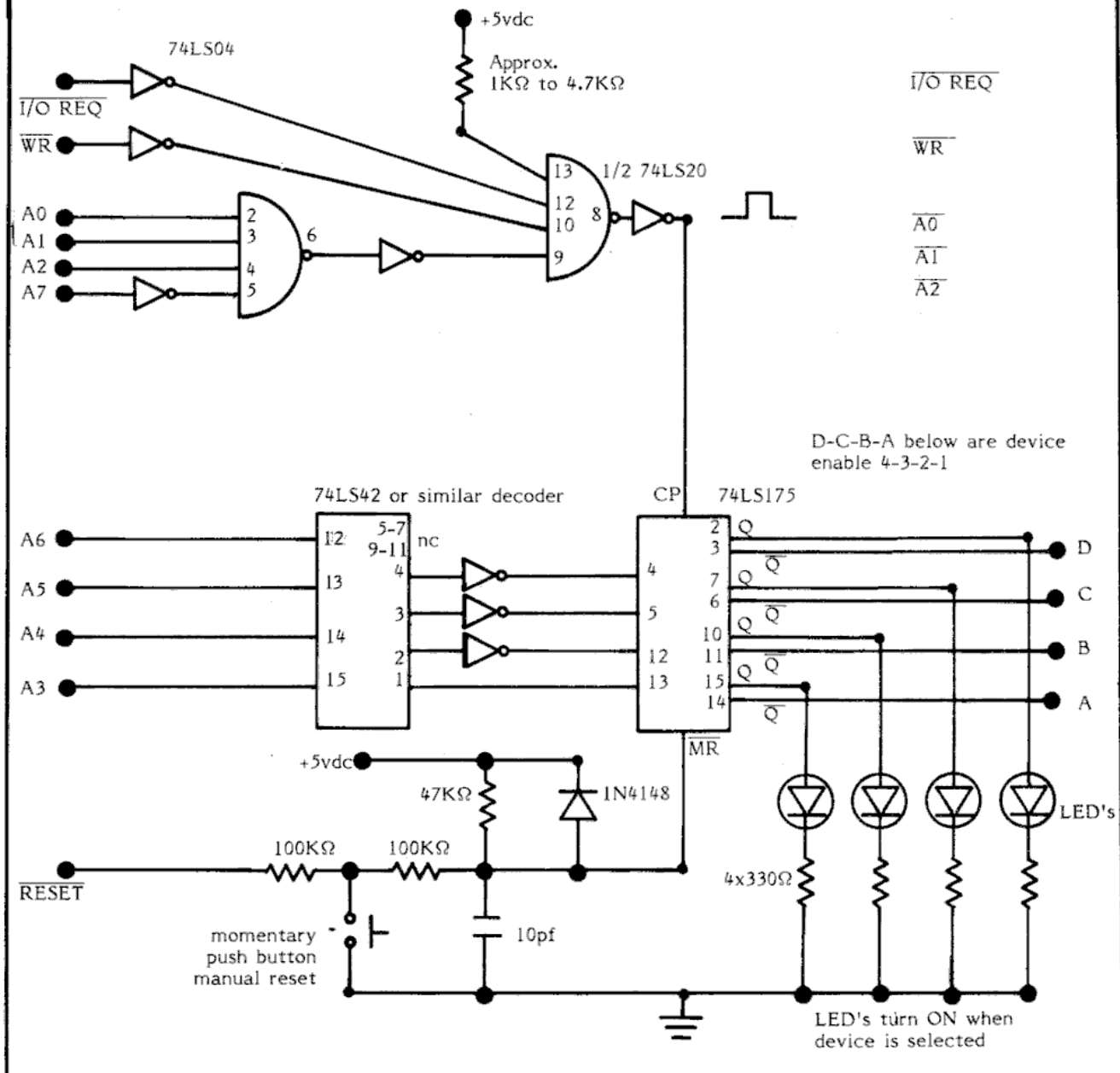
This project is based on a RADIO-SHACK 276-161 experimenter's IC board (\$2.95). With a wire-wrap edge connector and a small pc extender board, this makes an economical and reusable plug-in board for the ZX81/TS1000. Wire-wrap is recommended -- it's easy to undo and is often more reliable than soldering. The wire-wrap sockets you accumulate can be used over and over again with care. Although wire-wrap can be done with a manual step-by-step tool, something like a "Just-Wrap" daisy-chaining wire wrap tool by OK eases the burden considerably.

Figure 1. Using a Radio Shack Experimenter's board with a ZX81/TS1000



If you do not wish to make your own wire-wrap connector and pc extender board, they can be obtained from Paul Hunter, 1630 Forest Hills Drive, Okemos, MI 48864 for \$5 and \$1.50 respectively.

Figure 2. Paging Control Circuit



With such a tool, a project like the one described can be put together in less than an hour. The socket-wrap ID's by OK are useful if you're not used to looking at IC's from underneath. The Radio Shack experimenter's board stands up behind the ZX81 as shown in Figure 1. With the connector to the computer mounted flush to the board, and soldered or glued in place, there is plenty of room behind the board for the 3-level wire-wrap sockets. Be sure not to mount any components or sockets below the 16th row of holes from the bottom (the 0-0 line). Solder the pc edge connector in place between the wire-wrap pins of the socket after you have made connections to the pins.

To complete the prototype board, solder a 10 uF. tantalum capacitor between the +5V power supply pin and ground. Discrete components like LEDs, resistors, capacitors, etc. can be mounted on their own wire-wrap pins or on DIP headers and plugged into sockets.

THE CIRCUIT

The schematic for the paging control is shown in Figure 2. Note that no data lines are used -- the device select signals are derived from decoding the address lines alone. A0, A1, A2 are high and A7 low to avoid system conflicts. These lines, with the $\overline{I/O\ REQ}$ and the $\overline{WR}$ signals are used to generate a positive-going pulse which enables the 74LS175 latch. The four remaining address lines are used as inputs to the 74LS42. This decoder ensures that only one device is selected at a time. If you are confident enough and only have four devices from which to select, then the LS42 decoder can be omitted -- connect the address lines A3-A6 directly to the LS175 latch.

The choice of a 74LS175 latch allows an automatic reset on power-up. The outputs from the latch are arranged so that the default, or reset, condition is that for which only device number 1 is

The addresses used to operate the system are as follows:

ADDRESS decimal	A7	A6	A5	A4	A3	A2	A1	A0	74LS42 pin 1 2 3 4	Input to 74LS175	74LS175 pin 15 11 6 3	Output Q Q Q Q	RESULT
7	0	0	0	0	0	1	1	1	L H H H	L L L L	L H H H	RESET condition (device 1 on)	
15	0	0	0	0	1	1	1	1	H L H H	H H L L	H L H H	Device 2 enabled	
23	0	0	0	1	0	1	1	1	H H L H	H L H L	H H L H	Device 3 enabled	
31	0	0	0	1	1	1	1	1	H H H L	H L L H	H H H L	Device 4 enabled	

enabled. Upon power-up or reset you should observe LED1 on and all others off. When you test the board do so first without any peripherals attached.

The routine to select a particular device is simple enough;

```

CODE      Z80 MNEMONIC
211        OUT
N          address (7, 15, 23, or 31)
201        RETURN

```

The routine can be placed in a REM statement. For example, the first line in a program: 1 REM PEEK # TAN (# = space)

The device can be selected by: POKE 16515, (7, 15, 23, or 31)

and the routine executed by: RAND USR 16514

It remains, of course, to incorporate a device select signal (active low) in the selection of a particular peripheral device. Most peripherals are memory-mapped and are enabled by the MREQ control signal with some combination of RD or WR. If this is the case then the device select signal and MREQ can be gated together through a 74LS32 OR gate.

Veni, Vidi Video, Vici

So far in "The Custom T/S," I've shown you how to overcome problems with all three of the little inline jacks on the left side of your computer, with fixes that handle trouble with power input and tape-recorder functions. There's only one other connection (other than the rear edge-connector) left to cause any grief; the TV output. Actually, this one is the least likely of all four to be the cause for concern, so perhaps it's just as well that I left it for last. If you get a LOUSY picture, chances are that the problem is with your TV. If you try to use an older tube-type or hybrid (tubes + transistors) TV, you probably won't be too happy with what you see. I've had fine results with the Sanyo (also sold under the Sears brand-name) 12" solid-state model selling for around \$80. This one and others like it, by Panasonic, Samsung (the same company Sinclair Ltd. is currently dealing with), Tatung, and their various name-brands, probably represent the best buys for a "monitor" that will also double as a TV. For dedicated computer use though, you'd probably be happier with a nice green or amber-screen "computer monitor" since these can be found in about the same price range. You may have gotten a good TV picture with your basic machine/RAMpack only, but find that the picture gets dirtier as you add other peripherals. Unfortunately, there is no jack provided for a video monitor on the ZX81/TS1000. It won't work if you try to use the TV jack.

In this installment, I'll show you how to use a video monitor with your ZX/TS, and while we're at it, we'll clean up the signal with comparators and allow a switchable "reverse video" option (not included on most under-\$100 monitors). The cost is about \$10 in parts. This and the "clean-up" feature make the project worthwhile even if you want to keep using a TV (such as my small but very crisp Panasonic 3" portable).

This should take care of any remaining complaints you might have about your computer's video. (Except its speed in printing to the screen; which we can't do much about until we build John Olliger's video upgrade in the first two issues of Volume 2.)

But first, I'd like to present a "video primer" for beginning to intermediate ZX/TS computerists. So on the next pages you'll find "everything you wanted to know about TV, but didn't know who to ask." Hopefully this will give you the "why's" and motivate you to tackle the hardware project (the "hows.") If you're a software hacker, knowledge of the hardware of TV might help in your making sense of the ROM's TV display routines. The discussion is limited to monochrome (B&W) systems to keep it simple, as you lucky TS2068 users already have all you need for connection to a color monitor.

A Video Primer

TURNING A POINT INTO A LINE AND A LINE INTO A PLANE

A television set (or video monitor) is a serial device; this means that at any given time, only a single dot of light is illuminated (or not). The signal that reaches the TV is processed into a voltage called a "video signal," which increases or decreases to vary the brightness of the dot as it travels across the screen. The dot starts at the top left corner of the screen, traces to the right, then flies back to the left and traces the next line. This happens 15,750 times a second. Meanwhile, the dot is swept from top to bottom at the much slower rate of 60 times a second. When the spot reaches the bottom of the screen, it flies back to the top to start over.

So given these facts, we should be able to figure out the total number of lines, right? So let's divide 15750 by 60 and get... what? 262.5? That's right; only 262 and half lines are drawn every 1/60 of a second. The "one half" means that the spot is already half-way through the next horizontal line when it gets back to the top. The result is that the subsequent set of 262 lines is drawn interleaving the first set. Note that while there are 60 "frames" per second, only 30 complete pictures ("fields") are generated each second. This approach reduces flicker as compared to using a vertical retrace interval of 1/30 sec., since the eye can easily discern discrete "flashes" at that rate. So we have a total of 524 full lines, and a half-line at top and bottom, for a total of 525 lines, the quoted resolution for the NTSC (American standard) TV system.

This resolution figure assumes that the vertical return (retrace) happens immediately. Actually, this time is non-zero; the result is that some horizontal lines are lost during the retrace interval. Other lines are lost off the top and bottom to ensure that the borders will always be off-screen. Also, the top and bottom of the actual computer display is even less than the physical limit, for the same reason. The end result is that, in the ZX/TS system, only a total of 192 (24 X 8) lines are actually used.

Similarly, the horizontal retrace operation takes a non-zero time to complete, and again we aren't using the full 1/15750's of a second to actually convey the data in a line. Good thing, for us ZX/TS users, since it's during these "dead times" that your poor overworked machine has to do its actual computing in SLOW mode. With the American TV standard the ratio of "on time" to "off time" (as far as the screen is concerned) is about 6:1, which is why SLOW mode is 1/6 the speed of FAST mode. Yes, I said 1/6, even though your manual says "1/4." This much under-publicized fact was first brought out in SYNAPSE (the publication of the Central PA ZX/TS UG); those guys are really on the ball! The reason is that in the 50 Hz. PAL (British and European) TV system, there is more time to "do other things" (i.e., compute) during the retrace intervals. That bug in the manual has persisted to the present TS1500 manual! While we're on the subject, dear

North American reader, please note that any references you see to "50 Hz." or "50 times per second" in the manual or other books should be changed to "60." (e.g., anything having to do with PAUSE, FRAMES, etc).

But back to the topic. How does that little spot know exactly when to start a sweep? This is done by sending "synchronizing pulses" along with the stream of video information (light and dark). The width of these pulses determine the "dead-time" between adjacent lines (horizontally) and frames (vertically). These pulses are negative-going; i.e. "negative logic," which has the same advantages, engineering-wise, as negative logic in digital circuits. The level of the signal is typically about +2 volts, and drops to close to 0 volts during retrace. A circuit in the TV/ monitor called a "sync separator" sorts out these pulses and uses them to "lock-step" the horizontal and vertical oscillators which control the position of the electron beam (spot on the screen). To see what happens when you don't sync the video signal, mess up your horizontal or vertical "hold" controls. See how the sync signal "locks in" the picture over quite a wide range of adjustment. By the way, the horizontal hold works as a centering control on most sets.

Get the Picture

Now we've got a lit, stable screen ("raster"), but it's blank. So at the same time as this is going on, we have to vary the brightness of the spot to make a picture. This is done with the "video signal," which is exactly synchronized at the sending end (computer or TV camera) with the sync pulses. When the signal is as high as it will go (typically 1 volt) the spot is bright, and when it is zero, the spot is completely darkened. Note that the video signal uses "positive logic" in the NTSC video system.

By now you may be wondering how TV deals with the three different signals involved; horizontal sync, vertical sync, and video. The solution is to add them together and send them all as one signal. During sync pulses (retrace) we couldn't care about video, since we're "off-screen" anyway. So we might as well superimpose the video on top of the sync pulses. The sync signals are timed such that the vertical pulses is much wider than the horizontal ones. The result is a monochrome "composite video" signal, as shown in Figure 1. Note that the negative-going sync pulses help insure that the screen is black during retrace; another reason for this approach, since it makes the "retrace blanking" circuitry quite simple indeed. At the receiving end, the sync separator can easily differentiate the vertical (wide) pulses from the horizontal (narrow) ones and from the video signal (time between sync pulses).

So far the discussion has been general enough to apply to both TV's and monitors. What makes the difference between the two is that the monitor will accept a composite video signal, whereas the TV won't. In order to get a signal to the TV without wires, good ole AM radio is called "into the picture." That's right, AM. The composite video signal is "modulated" onto a radio-frequency (RF) "carrier." The composite signal simply varies "how big" (amplitude) the radio signal is at any given time. The "modulator" in your computer is actually a tiny TV transmitter, consisting of an oscillator (about 55-65 MHz. for N.A. channel 2-3 units) and the modulator itself which controls the radio signal's amplitude. The signal then is attenuated (reduced) so it "looks like" something a TV would expect at its antenna terminals (about 10E-7 volts). At the TV end, the signal is amplified (RF amp), reduced to a standard, more manageable frequency (converter), amplified some more (IF amp), and only then converted back to a video signal (demodulator). With all that electronic "stuff" to go through, no wonder TV displays don't look as good as video monitors, which accept the composite video signal directly.

At this point, I might mention that the sound portion of the NTSC standard signal is FM (frequency modulated), unlike the video portion. This is why it is difficult to implement "through-the-TV" sound on any computer; the ones that do (like game machines, CoCo, and others) usually don't sound too good because they're actually using a bastardized AM signal to try to "trick" the FM demodulator into a reasonable facsimile of the intended sound. Once again, you TS2068 users have the edge with your direct audio-output jack and built-in speaker.

So a TV is nothing more than a video monitor with a radio receiver (a rather complicated one, but still a radio) attached to its input. So how about tapping into the TV's composite video line directly? Sounds good, but rots a ruck. On most TVs you'll have trouble right off the bat because the TV is not isolated from the AC line with a transformer. At best (if your system is grounded) you'll have blown fuses and sparks, at worst (if using a "floating ground" like most) you'll have a serious shock hazard. Well, yes, you can isolate the two systems using an opto-coupler or isolating power transformer. Even so, it might be hard to locate the composite video line; and if you do, it may be offset by some arbitrary voltage. The goal of TV designers is to make a working set with the lowest parts count, and over the years they have come up with some remarkably tricky but effective ways to make a bunch of hardware do "more than it should." The "cost" is that it makes things difficult for experimenters. Even on portables that are isolated, there may not be an accessible point to inject the composite video; that point exists, of course, but it might be buried inside some chip where you can't get to. Trust me! You'll be better off getting a monitor made for that purpose. But if your budget doesn't allow this additional piece of hardware, there is still something you can do to substantially improve your TV computer display.

The signals that race around in your computer are digital; i.e., "square waves." They have a rapid ON transition and a rapid OFF transition. If you've studied Fourier analysis, you know that rapid transitions in a waveform means a large high-frequency content (harmonics) in the waveform. If you haven't, don't worry about it; take it as "Blind Faith Principle No. N+1." These harmonics extend well into the VHF range (especially into the "low VHF" area our computers work in). This is what causes those annoying moving Moire-pattern effects; the more hardware you add onto the rear-connector, the more your system acts as an antenna to radiate this garbage to your TV.

This type of interference can be greatly reduced by changing from a VHF (channel 2 or 3) to UHF (typically channel 33) modulator. UHF is so much higher in frequency that the harmonics from the computer have a negligible effect. Such a modulator can be purchased from Computer Continuum and others, and simply replaces your existing modulator. Add our clean-up/reverse circuit, and you can get most of the advantages of a monitor using a good TV.

TVs are too Flashy

There is another good reason to switch to a monitor, dealing with the persistence of vision and the persistence of "phosphors." The light-producing coating on the inside of a picture tube actually continues glowing for a while, just like "glow in the dark" substances (essentially the same thing). On TV sets, the phosphors used have an intentionally fast "decay time" so that there is no or little "streaking" of fast-moving scenes. The price is that there can be noticeable flashes or flicker, especially when you're shifting your glance from the screen to somewhere else (like a program listing) and back. Computer monitors use a picture tube with a longer-persistence phosphor; the image glows for a longer time, and almost completely eliminates flicker or "strobing" with fluorescent lights. This is great for programming, text, etc. but a mixed blessing in other ways. For one thing, it causes noticeable "trails" on fast-moving graphics (particularly if the moving "object" is light on a dark background) which may or may not annoy you while playing space games. The other minor negative effect is that it makes light-pens more difficult to adjust, since they depend on receiving a sharp pulse when the light-dot passes the location of the pen. These points are almost trivial, though, when compared to the reduced optical annoyance when looking to the screen and away again for hours on end.

Getting a Monitor

So what's a good monitor to get? Probably the most common right now is a unit brand-named BMC which is currently appearing at stereo/video outlets. It's not bad, and produces a bright, green picture with adequate horizontal resolution. It does have a minor "bug" when used with the ZX/TS which I personally find objectionable; there is a pronounced vertical bounce when the computer comes back from FAST mode or after a PAUSE. This unit (and a few others) is essentially a re-worked TV; early units even still have the speaker grille in the casing. A TV is not meant to have the video signal, including retrace pulses, disappear and re-appear. I use a 9" USI green-screen (excellent product) and have used the Zenith 12" amber-screen (prettier case but more expensive) on the battery-evaluator project for G&M Power Products. Both provide near-instant stability on return from FAST, though the USI is just a tad quicker. USI also makes a 12" version which includes a reversing switch, and both the 9" and the 12" have a DC restoration switch (discussed later).

Don't pay too much for a monitor! Spend a few hours at a book stand looking through some magazines to find the bargains. You shouldn't pay more than \$100 for a decent green-screen, slightly more for amber. (I prefer the green, myself, but this is largely a matter of taste; personally, I seriously question the claim that amber causes less eye-strain.)

Now, let's take a look at the question of "reverse video." I've found that most beginning computerists prefer the black letters on white as provided by the ZX81 and other home machines. Perhaps this is because it is most like dark ink on white paper. However, the longer you stare into the screen debugging programs or entering text, the more you'll appreciate light letters against a black background; it's simply easier to look at for long periods of time. You don't have all that light in the background burning a roughly rectangular image into your retinas. I also like what happens during FAST mode; instead of going grey, the screen simply disappears (goes black). The "flash" is not nearly as pronounced, and the effect is more like "I'm going away to think for a while" than "I've just lost my marbles."

Another worthwhile feature is a high-impedance input option, provided as a switch on some models (incl. USI). It takes less power to drive a hi-Z input, so the "driver" you make for it can use a lower-power transistor (and takes less from the supply).

Finally, look for a "DC Restoration" switch. You may have noticed that night-scenes on TV don't look black, they're grey. This is because video amplifiers are usually AC-coupled, which means that the black-level varies with the ratio of light-to-dark across the screen. A circuit called a DC restorer is used in the better monitors to clamp the black-level at or near "true zero" no matter how much or little is printed on the screen.

A VIDEO REVERSE BOARD

(or, "How To Drive a Monitor")

The easiest way to connect your computer to a monitor is to simply take the composite video signal from pin 16 of the ULA ("SCL" chip). Since this isn't capable of much power delivery, it will only work in the "high-impedance" input setting. If your monitor doesn't have a Hi/Lo-Z switch, open the monitor and trace where the input goes on the circuit board. Nearby, you'll find a 75-ohm resistor (purple-green-black- gold or silver). Simply remove this resistor and you now have a Hi-Z monitor.

A better approach is to isolate the input using a single transistor "voltage follower" as in Figure 2a. Connect the base to ULA pin 16, the collector to +5V through a small resistor, and the emitter to the monitor. The monitor's own 75-ohm terminator resistor acts as the return to ground. You can get away with a smaller transistor (2N3904, etc.) if you change this load resistor to a higher value, say 330 ohms. If you want to keep the 75-ohm input, then use a medium-power transistor like the Radio Shack 276-2030 (same one we used for the motor mod last issue). The collector resistor is to limit the maximum current if you should short the output. Its value should be about 1/4 the value of the emitter load resistor. Alternately, omit the collector resistor and AC couple the output with a 47uF. cap. However, on some monitors this will degrade DC Restoration (brightness stability of background), so generally the DC-coupled approach is preferred.

Figure 2A. Video Driver

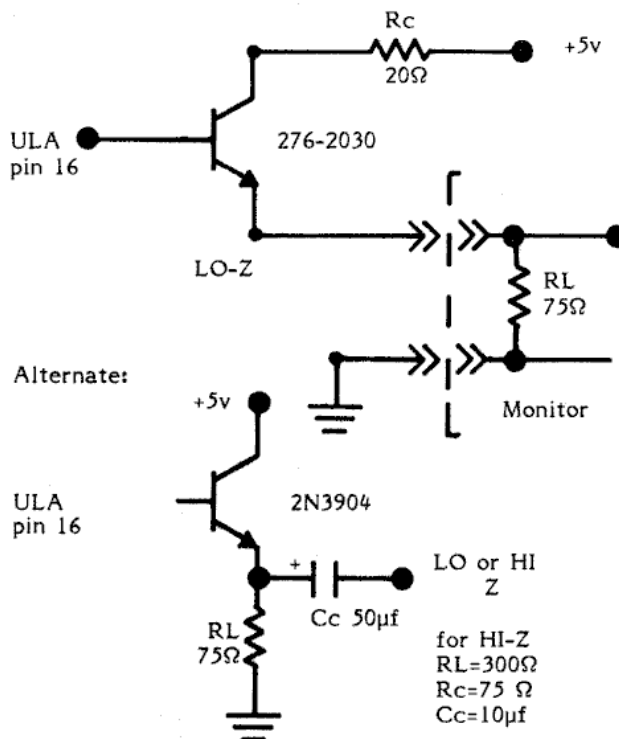


Figure 2B. Video cleanup/reverse circuit

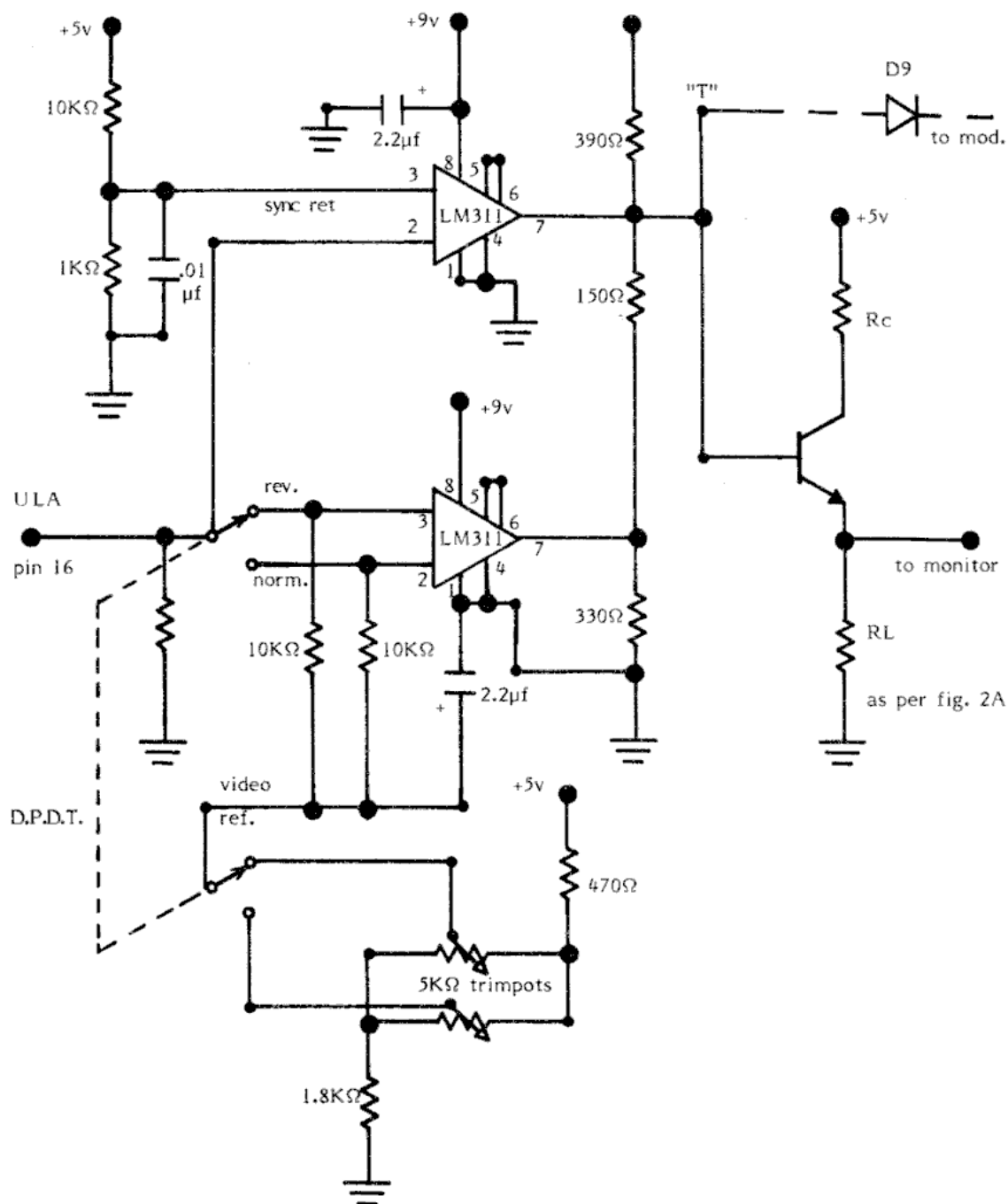
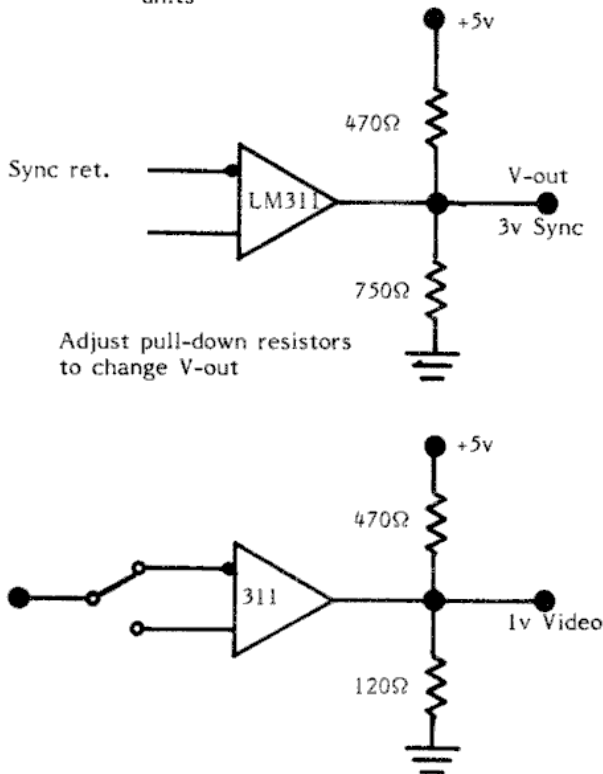


Figure 2b shows a simple but very effective video cleanup and reverse circuit. It's the fourth and simplest version of this that I've come up with. It gives you switchable normal and reverse, and cleans up the video signal in the process. Although we get rid of the VHF fuzzies, the video signal itself usually has garbage riding along. This shows up as tenuous vertical lines on a white screen, and since they don't move around they're much more tolerable than the VHF harmonics. But while we're at it, let's get rid of them. The lower comparator does this for us; its reference voltage is set to the midpoint of the video signal, adjustable by the little trim pots. Depending which input the video

goes into, we get either normal or inverted video output. Separate adjustments are provided, so you can trim up each mode separately for the best picture.

The Vref of the top comparator is set to a low value so it responds only to the sync pulses. (Note that we couldn't just invert the entire signal to get reverse video, since the sync pulses must be negative-going.) So, this comparator is our "sync separator." The outputs from the two are combined in a resistor chain to give about 2V black level and 1V video. This goes to the emitter-follower, thence to the monitor. It works great!

Figure 2C. Output circuit for split video/sync units



Build this up on as small a perf board as you can fit everything onto, and keep all lines as short as possible. Mount the switch (e.g. a miniature toggle) close by, preferably on the board itself. Open the anode end of diode D9 on the ZX81 board (goes to ULA pin 16). Connect pin 16 to the Vidrev input, and anode D9 to the output marked "T." This

way it will also provide reversing for the TV. Try bypassing D9 entirely; on some machines it looks a little better. Connect +5 and +9 at the nearest available point. Keep the ground return (to modulator case) short! The LM311 comparator (an old standby) is very klutz-friendly, so if you have an error, it usually isn't catastrophic.

When you get it working, you'll probably find that your picture is small, with big wide borders. This is easy to fix. Vertically, simply adjust the "Vert. size" control on the back. You'll have to touch up "Vert. Lin(earity)," which adjusts out any line compression at top or bottom. These two are also interactive with "V. Hold," so it might take a while till you get it just right. Horizontally, you usually have to remove the back cover, and locate a little coil marked "H. Width." It has a ferrite slug in it. Adjusting this counterclockwise (out of the form) widens the picture. On every monitor I've played with, the best picture is the widest picture; i.e., with the slug completely removed. The reason you had to make these size adjustments is that most computers use much more of the available screen-time than NTSC TV; i.e., the sync pulses (retrace time) are narrower, and better resolution results. Remember, we're only using 192 out of a (theoretical) maximum of 525 lines, whereas other computers may print up to 320 lines; the factory assumes you'll change it as required, and there are usually no "warranty seals" to indicate that you've opened it to align it to your machine.

To center the picture, adjust the little tabs that rotate the metal rings on the "front" of the deflection yoke assembly on the neck of the CRT (picture tube). These are circular magnets which will allow you to center the picture. They don't quite work the way you'd expect at first, but you'll quickly get the hang of it. Try using a mirror to see the screen while you're experimenting for "the ultimate picture" from the back.

VDAQ - A Data Acquisition Development Program

BY FRED NACHBAUR

Here is a program that demonstrates a simple yet effective way to get analog information (data) into your ZX81/TS1000/TS1500 computer, with a minimum of additional hardware. Also included are various plot and display routines so you can get visual and printed output of your acquired data. Along the way it will show you various "tricks" and other tidbits that might help out in your programming.

A little background is in order. My original intent was to write a "universal" data acquisition program for the popular "VOTEM" analog interface. "Universal" is a pretty broad term, though; it didn't take me long to realize that a truly general program would be difficult to implement in 16K. A better approach is to write a number of customized programs, each tailored for a certain range of applications. If nothing else, possibilities for display and printer formats are almost endless, not

to mention ways in which the data may be processed once acquired. So what I'll try to do in this series of articles is to present some "basics" from which you can expand to suit your application.

In this issue I present a complete program which will sample data at a rate from about 1 sample per sec. to about 350 samples per second, with resolution between 8 bits (fastest sample rate) and 14 bits (slowest sample rate.) It can store 2048 discrete samples, each one consisting of two bytes. Data is organized into "screens" of 64 samples. Each screen can be acquired/ displayed individually, or several screens can be chained for longer time-bases. As listed, the capacity is 32 screens, but you can easily make this more or less with simple changes. TS2040 printer is supported, and you can even do "side-winder" plots with a vertical resolution of 60 points and length (width?) depending only on the number of screens plotted. Plot limits are easily changed, and screen plots are done in machine-code for speed. Since this program

is optimized for speed, no data display occurs during acquisition, and there are no additional pause routines for longer time-bases. This will be the topic for the next issue. But for now, let's backtrack a little and look at how the hardware works, and then at what you'll need, hardware-wise, to make the software work.

Theory

There are several ways to get analog data into a computer. The usual approach is to use an analog-to-digital converter; such devices have at least one analog input channel, and a digital data bus for output. The binary number appearing at the data bus is proportional to the voltage at the analog input. Converters are available to give a resolution anywhere between 8 and 14 bits (15 bit with a minus sign is present "state of the art"; that's 4 significant figures. ed.), depending on the accuracy required. Eight-bit converters are relatively easy to implement on an eight-bit CPU like the Z80, since the data can be impressed directly onto the CPU's data bus. For higher-resolution A-D's the acquisition has to be done in two "chunks," using some decoding scheme to first look at the first 8 bits of the output bus, then at the remaining bits. In either case, the page to start a conversion can be either I/O or memory-mapped. The resulting data is then loaded into memory for subsequent display and other processing.

With all its advantages (mainly speed) this approach does have a few drawbacks. First among these is that it requires connection to the computer's data bus (8 lines) as well as to at least part of the address and control busses (lines like MREQ, IORQ, RAMCS, etc). Another drawback is that matters start getting quite complicated if greater than 8-bit resolution is required. Because of all the circuitry needed, most folks would prefer simply buying a commercial module rather than mess with all those wires, decoding, timing circuits, etc. See the Ener-Z Report Generator review elsewhere in this issue for an overview of one such package. Another is the Computer Continuum "(8+8)*8" A-D, D-A board (reviewed in a future issue.)

There is another route we can take if high speed (faster than about 300 per second on the ZX-TS) is not of the essence. It is an old method, but still very useful. In this method, the voltage is first converted into a stream of on-off pulses whose frequency is proportional to the input voltage. Such a device is called, aptly enough, a "voltage-to-frequency converter," or more simply a "V-F." Now all that is required is a single input bit to get the stream of pulses into the computer. It is a serial rather than parallel approach.

Your ZX/TS has a dedicated input port already - the EAR jack. A single Z80 IN A,(FE) command gets the state of the EAR jack input in bit 7 (the most significant bit;) from there it is a simple matter to write software to count how many pulses (state changes) appear during a fixed time span - the resulting count will be proportional to the input voltage. By appropriately modifying the IN command, this approach will work with any computer with an IN

port such as a cassette input. Calibration can be done entirely in software, making the hardware aspect simple indeed.

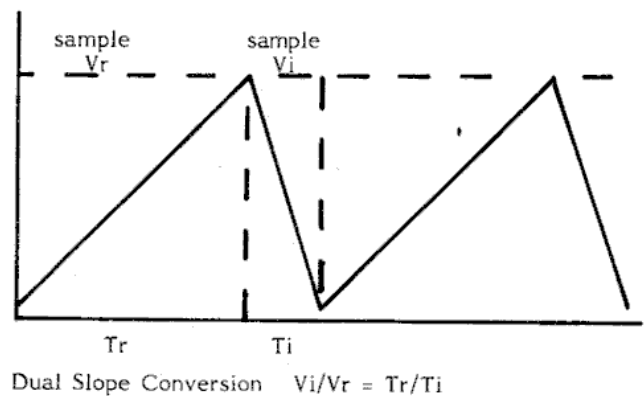
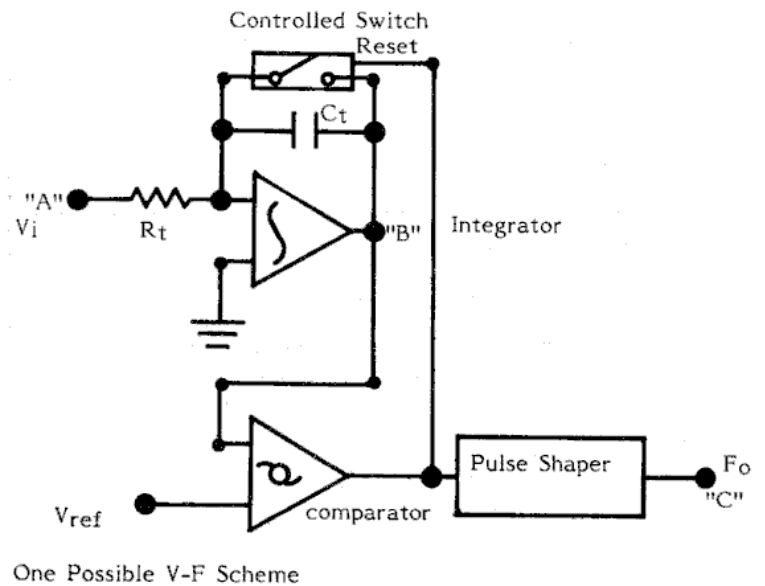
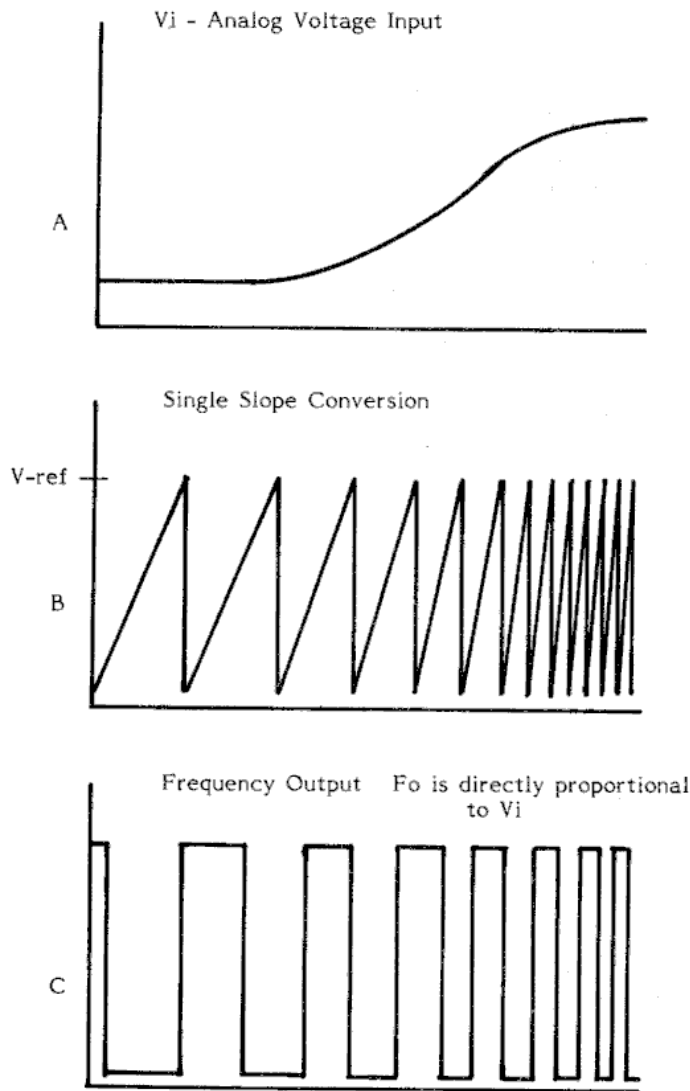
V-F converters almost always use a circuit called an "integrator," which provides an output voltage proportional to the integral of the input voltage. This means that the higher the voltage to the input, the faster the output from the integrator will ramp up (increase). This signal is applied to a comparator, and when it exceeds the comparators upper trip point the comparator toggles (changes state), at the same time resetting the integrator. Result - the higher the input voltage, the higher the frequency of the pulses at the output of the comparator. See Figure 1. This is called "single-slope V-F conversion." Also possible is what is termed "dual-slope conversion," in which the conversion is done in two steps; a known reference voltage is used to ramp the integrator up, and the unknown voltage is used to ramp the signal back down. The ratio of the ramp-down time to the ramp-up time equals the ratio of the reference voltage to the unknown voltage, from which the unknown voltage can be easily derived. The dual-slope approach is more immune to drift and other inaccuracies, and allows "auto-calibration" in dedicated devices like the ubiquitous Digital Volt Meter (DVM). (Virtually all inexpensive DVM's use some variant on this approach - as I mentioned earlier, we're not dealing with anything new and esoteric here.)

When using V-F data acquisition, accuracy depends on the "acquisition interval." This is how long we are counting, and therefore how many counts we read for a given input voltage. If "full-scale" (the highest voltage we wish to measure) only returns 256 counts, we get 8-bit accuracy; but if it returns 16384 counts at full scale we have 14-bit accuracy. So you see that the more accurate you want it, the longer you'll have to sample the input pulses. How fast you can go depends on computer speed; on the ZX-TS it takes about 1 second to get maximum resolution (better than 14 bit, or about .005%). At the other extreme, we can get 8-bit resolution (about .5%) with acquisition times down to a few milliseconds. In this program we'll experiment with the relationship between speed and accuracy by making the acquisition interval equal to the sampling interval; as soon as one sample is acquired, the program goes right on to the next, with no additional timing pauses. Timing is varied by changing the acquisition interval; as a result, the full-scale count and the number of counts per volt changes with sample-rate, and has to be compensated for when converting the counts into voltage or other units.

Next time we'll add programmable pauses so the sampling interval is greater than the acquisition interval for longer time-bases (minutes, hours, days). This will also allow time to update a continuous display during acquisition. You are encouraged to customize and improve these programs from there, to come up with exactly what you need for your application.

What applications can it be put to? Well, just briefly mentioning some of the possible input devices should help suggest plenty of ideas. You could use a potentiometer for position sensing (simple robotics, wind or antenna rotor direction

FIGURE 1 - PRINCIPLES OF V-F DATA ACQUISITION



indication, etc.), a photocell (light meter, etc.) or a strain transducer (for forces, stresses.) Use a microphone as the heart of a decibel (sound level) meter. Let's not forget about temperature probes, for clinical/ meteorological "thermographs," solar heating control, or heat-transfer studies. For some uses you won't even need the V-F. The obvious example is as a frequency counter (this program will handle frequencies from about 2 Hz to about 20 kHz.) Similarly, you won't need the V-F for such things as radiation monitors (just run a Geiger counter output to the EAR jack and have the computer count the pulses. Use Tom's moving-trend routine to smooth the data as required.)

A final point before we get on to brass tacks is that the V-F approach lends itself very well to remote sensing; you can locate the V-F with the sensing device, and bring the data to the computer along a single digital line (use a comparator for cleanup at the receiving end if needed.) This is

much more accurate and free of glitches than running a long analog line, and is cheaper and easier than bringing the data in on parallel data + control lines. But now that I've firmly convinced you that "you NEED this program," let's take care of the hardware aspect, and then briefly go over the system software (the fun part).

The Hardware

If you have a ZX/TS with 16K RAM and a VOTEM analog interface, you already have all the hardware you need. If you don't have a VOTEM, you should consider purchasing either a complete or partial kit (any combination of bare board, documentation and V-F chip). (they are still available. ed.) Down East Computers' prices are not out of line, and their unit is the best approach if you're not willing to build a V-F board from scratch.

For those of you who don't mind a little hardware hacking, I'm printing a "rip-off" circuit for the LM331 V-F chip, derived from the National Semiconductor application notes for this device. It is accurate, inexpensive, and quite readily obtainable (if you can't find it at your local jobbers', check with the nearest location of Avnet/Hamilton.) Values shown are for a 1 volt full-scale sensitivity, you can easily sense higher ranges by preceding the V-F with a resistive voltage divider. The absolute precision of the other parts is non-critical, since we'll calibrate the final circuit anyway (using a software scale constant). What IS important is that the timing parts be stable with temperature; the timing capacitor should be polystyrene or polycarbonate, and the resistors should be metal-film. Avoid extreme temperature changes around the V-F board if possible. Wiring is non-critical, and you can use wire-wrap on a perf board to do the actual construction. You can power the V-F from the computer supply, or use a small battery (especially recommended if you'll be locating the interface remotely.)

You can use the Radio Shack 276-1790 V-F, F-V IC. Applications notes are provided with the part. (Though we won't be using it, this unit also allows you to convert from a frequency back to a voltage - you could conceivably work up a circuit which will translate a frequency - such as you can make available at the MIC jack- into an analog voltage, which can be used to control such things as proportional heaters, motors, etc. More about this in a future installment.)

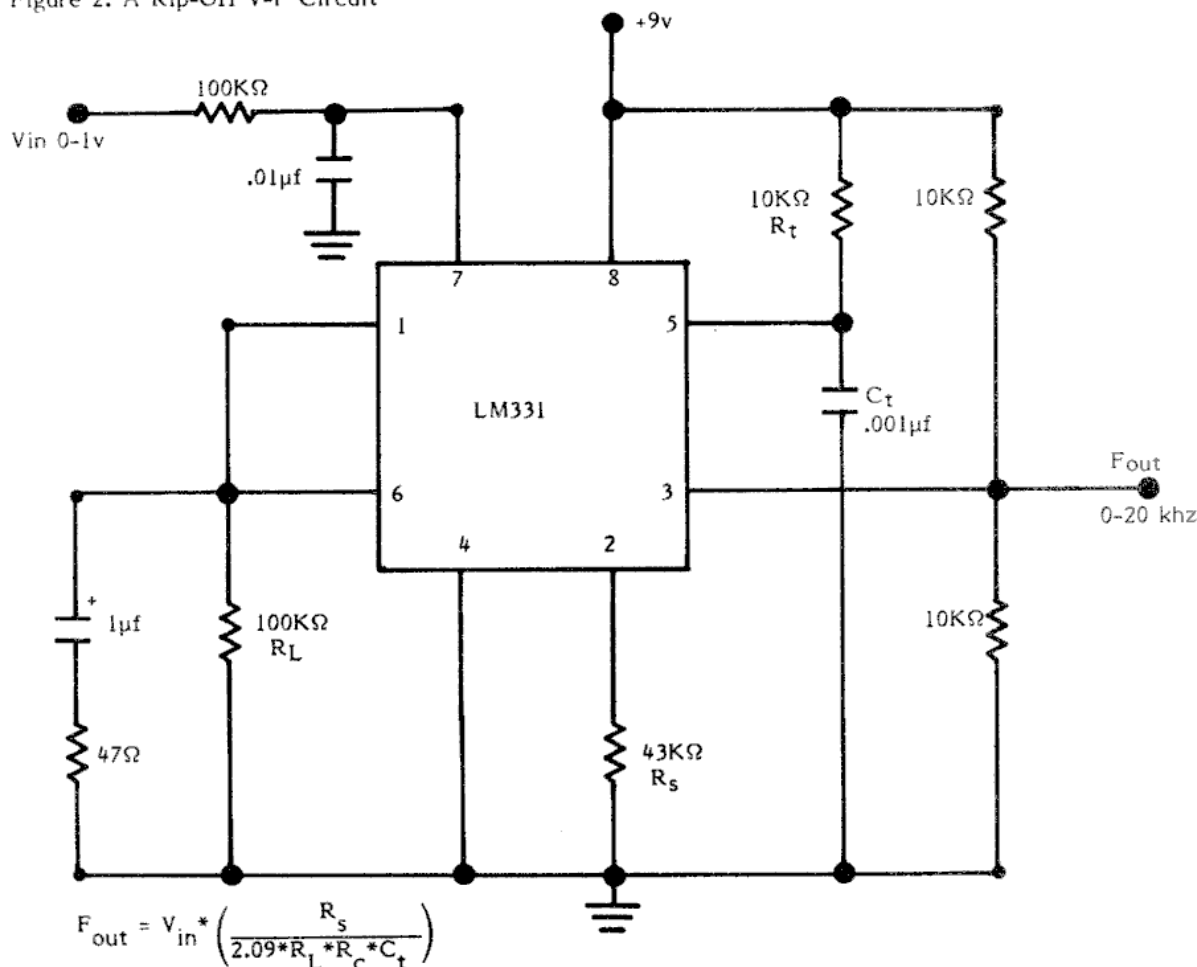
If you're using a VOTEM, you should disconnect capacitor C8 at the input to the V-F, or replace it with a much smaller unit (e.g. .01 uF), otherwise you'll get considerable waveform distortion due to its filtering effect. (A 20 Hz. square wave won't look the least bit "square.")

Whatever V-F circuit you use, the maximum output frequency should be limited to about 25 kHz. At higher frequencies you'll rapidly get inaccuracy due to "strobing" effects, since the rate at which the sampling loop runs is about 65 kHz. The amplitude requirement is that the "on" voltage should be 2-5 volts, and the "off" voltage should be a few tenths of a volt maximum. Duty cycle (ratio of On-time to total cycle time) is non-critical, but shouldn't deviate too far from 50% for best results. Run the output to the EAR jack, and the hardware and interfacing is complete.

The Software

In order to be able to present the entire program in one installment, I'll have to gloss over some "fine points" regarding the program this time. This is perhaps just as well, since what you learn on your own is much more likely to stick than what you read. Instead, I'll concentrate on how to get the program entered into your computer, and on how to use it once it's debugged and safely on tape. Thus you'll have something to play with while we're working up the additions and improvements for the

Figure 2. A Rip-Off V-F Circuit



next episode. You shouldn't have much difficulty entering and debugging it, and you can use any of several different ways to enter the machine-code portion.

The Machine Code

First you have to enter the machine-code. Start with the usual line 1 REM statement, 407 bytes long. Use a toolkit or enter it the "hard way," one character at a time. Another way of doing it is shown in the listing for line 1 in the BASIC listing on page XX. Enter 1 PRINT 0 followed by 50 repetitions of +0. The computer interprets each zero as a floating point number - so it uses up six additional bytes for the binary representation of each zero. The first 0 takes 7 bytes, each +0 takes 8 bytes. We're using the ZX's extravagant number storage to our advantage, requiring only 101 keystrokes to enter 407 bytes. Note that this won't work using 1 REM 0+0+... because after a REM, the zeros are interpreted as characters (1 byte each). So use PRINT, then change the PRINT to a REM using POKE 16513,234.

However you get the REM statement in there, you can check that you have the right length by entering PRINT PEEK 16921, which should return 118 (end of line marker). If you want to make line 1 more resistant to accidental deletion, POKE 16510,0 to assign it a line number of zero. Now enter a decimal loader as on page XX, setting the counter limits (in line 10) FOR A=16514 TO 16843, and use the values shown in the decimal table. Or use a hex loader or assembler and enter the program from the Hot-Z listings. The locations from 41CCh (16844d) to 41D7h (16855) are machine-code variables, and need not be entered. Similarly, the 64 bytes at 41D8-4217h (16856-16920d) represent a machine-code "array", which holds the "timing value" for each screen. If you want more than 32 screens, make the original REM longer - e.g. for 40 screens, add two more "+0"s. (16 bytes.) This timing value is the number that is loaded into 409C-409Dh (16551/2d) in the acquisition routine, which determines how many times the program loops to take another look at the state of the input bit 7, and sets our acquisition interval or "window."

The first three routines are nested loops; MULTISCREEN calls ACQUIRE A SCREEN, which calls ACQUIRE A POINT. You could save some time between samples to improve accuracy as low timing values by writing it as one routine, but I left it as shown to make it a little easier to follow. The acquisition routine is a slight variation on the routine supplied with VOTEM. The next routine gets the start address of the specified screen being written to/read from. The data is stored in the first BASIC variable, D\$(4096), which has to be twice as long as the total number of points. Although the array is actually organized as 32 screens of 64 2-byte points, DIM D\$(32,64,2) or DIM D\$(32,128) will not work with this storage system, D\$ must be single-dimensioned.

The next routine draws the axes MUCH faster than any BASIC loop could, especially for the vertical axis. The rest of the code is the data graphing program, consisting of an arithmetic package (so we can do 16-bit by 16-bit division)

which is called by the plotting routine to evaluate each Y-position. The arithmetic package is a portion of the run-time package of Bob Berch's ZX COMPILER, and is actually more than we need here, as it also contains multiplication, ABS, SGN and other functions. It was used here to demonstrate that a compiler (especially Bob Berch's, of Cinagro Software, 19 Jacques St., Rochester, NY 14620) is a very worthwhile thing to buy even if you're more interested in hand-assembly; you can find many useful routines in the run-time package that will save you time in assembly by giving you function utilities to call when needed. Stay tuned to SWN for a significant article by Bob which shows how you can drive a serial printer from your MIC jack - meanwhile, make it worth his while to submit this article (and otherwise keep supporting your machine) by ordering his Compiler or other programs.

If you wish to get graphs dumped to your Centronics printer via Memotech CIF, you have to enter the print buffer and conversion table as per SWN 1:3, then enter the machine-code of this program into a line 2 REM statement. It will be displaced by 348d bytes, starting at 16862. You will have to change (offset by 348) all CALLs and references to machine-code variables or use Hot-Z II to relocate it for you. You'll also have to change routine addresses in the BASIC, and add the screen-dump given in 1:4 to replace the COPY command.

The BASIC

Once the machine-code is in, the BASIC should present no special problems. You might want to rearrange the menu options - e.g. put "Initialization" as option 1 and start it at line 1000, put "Acquire a Screen" at option 2 and start at 2000, or however you want it. Though the BASIC listing is only 2 pages long, there's quite a lot going on here, including some "frills" you could do without (like the pseudo-INPUT routine at 8000, which inputs 2-digit numbers without pressing Enter, or 1 digit numbers using Enter).

When using RUN for the first time (after SAVEing in case you have bugs in the machine-code), the program goes to the initialization/ calibration part of the program. You have to calibrate first voltage (calibrating the V-F) and then the time-base (calibrating the computer). Use a voltage close to full-scale, and monitor it with an accurate DVM while pressing Enter. The reference to "switches 3,6 and 7" refer to VOTEM, in which this connection measures the 1V reference built into the V-F. The display will disappear for about 1 second, then input the measured reference voltage. The program prints the observed count and counts per volt (at the longest timing value possible). Then you have to time how long it takes to acquire a screen of 64 points; it will be between 60 and 66 seconds, depending on the speed of your particular machine. Enter the actual time, and the computer is calibrated for time-base.

When acquiring data, you have three options for starting the acquisition; by pressing a key (option "K"), when the input frequency becomes non-zero (press "N") or when the input frequency goes to zero (press "Z"). You may have other requirements, which you should be able to program if you study how these options work.

Operation of the rest of the features is pretty straight-forward, and the prompts should help. When doing a side-winder plot, you might save paper if you first do a "dry run" (prints to screen only) to make sure your data doesn't go off scale.

Multi-screen acquisition and display asks first which screen to start at, then how many screens the acquisition is to run. Obviously you can't acquire more screens than you have available after the starting screen. The program then loads the timing value into the timing value machine-code array for each screen involved. (Assures correct scaling for each screen when plotting and printing later.)

Line 9999 shows an interesting option to the STOP command, first reported in SYNAPSE (Central PA Users' Group.) Look ma, no report code! Verify that the program indeed has stopped by pressing Enter or LISTing.

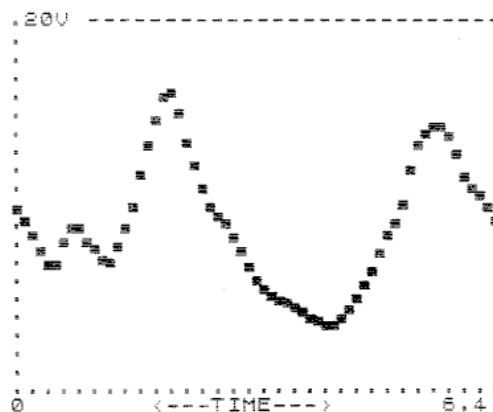
Another tidbit is demonstrated in lines 2090-2210 and a couple other places. S1 and S2 are the two bytes of the SEED variable which is set by RAND. This is a great way to convert a decimal number (e.g. R) into two-byte form; just RAND R, and then PEEK 16434 and 16435 to get the high and low bytes for that number. One warning - it doesn't work with zero, since RAND 0 means the same as RAND - i.e. get the value currently in the FRAMES variable. So for the low-count variable CL (count corresponding to the bottom of the screen) we have to use another approach (lines 2120-2130), as this variable may be zero. I first saw this programming device in Ralph Coletti's newsletter (no longer being published,) though it has also appeared in a couple other publications. Lines 4-7 restore the program listing cursor and the top line in automatic listings to line 8, in case you list line 1 and the automatic listing gets stuck. These POKes can be done in the immediate mode also.

When printing the data values, the program takes the acquisition interval into account and shows you only as many decimal places as you can expect for that acquisition rate.

You may wish to change the display format; this routine prints a two-column listing specifically with the TS2040 in mind, but is a little awkward for "screen-only" data display. Likewise, you might come up with a faster side-plot approach, perhaps using a m/c routine similar to the single-screen plot.

Other challenges: - write additional lines in the printing routine to compensate for the non-zero time between acquisitions. Modify the program to work with Nissim Elmaleh's Software-only Hi-Res or Russell's HRP hi-res printer package, or with Memotech HRG or other hi-res hardware package. Translate the program for use with the TS2088 - shouldn't give much trouble, though upgrading the display routines to hi-res might give you a challenge. Work out the machine-code to plot quantities with negative slope factors (such as most temperature probes.) Above all, have fun with this! More later.

Sample Run



FILE NO. 1
TIME BASE= 6.4 SEC.
SAMPLE RATE= 100 MSEC.

NO.:	MSEC.:	VOLTAGE:	NO.:	MSEC.:	VOLTAGE:
0001	100	0.0000	033	3300	0.40
0002	200	0.0000	034	3400	0.00
0003	300	0.0000	035	3500	0.00
0004	400	0.0000	036	3600	4.7
0005	500	0.0004	037	3700	4.53
0006	600	0.0003	038	3800	4.10
0007	700	0.0010	039	3900	0.00
0008	800	0.0003	040	4000	0.70
0009	900	0.0003	041	4100	0.40
0010	1000	0.0014	042	4200	0.00
0011	1100	0.7703	043	4300	0.00
0012	1200	0.7703	044	4400	4.30
0013	1300	0.7703	045	4500	0.03
0014	1400	0.7701	046	4600	0.70
0015	1500	0.0003	047	4700	4.40
0016	1600	0.0000	048	4800	0.00
0017	1700	11.770	049	4900	0.00
0018	1800	13.034	050	5000	0.16
0019	1900	14.770	0501	0100	0.10
0020	2000	16.001	0502	0200	0.00
0021	2100	16.004	0503	0300	10.41
0022	2200	16.11	0504	0400	14.00
0023	2300	13.40	0505	0500	14.00
0024	2400	10.10	0506	0600	14.07
0025	2500	0.000	0507	0700	10.00
0026	2600	0.000	0508	0800	10.00
0027	2700	0.44	0509	0900	11.03
0028	2800	0.1	0510	1000	10.00
0029	2900	0.40	0511	1100	10.00
0030	3000	0.03	0512	1200	0.07
0031	3100	0.70	0513	1300	0.00
0032	3200	0.04	0514	1400	0.00

Some Important Variables

Here are some of the crucial variables in the program. Some (like R, CL, C0) are also machine code variables: the address is shown where applicable.

VREF	full scale calibration value
C	counts at VREF. C0=65535 (later a tem. var.)
CL	counts per volt at C0=65535
C0	(16540-41) Timing value (number of acquisition loops). C0=65535*TB/TREF. Stored in C0 array for each screen.
TREF	Time required to acquire 1 screen at C0=65535
TB	Time-base for 64 acquisitions (1 screen)
SR	Sample Rate (mSec.)=1000*TB/64
CM	counts per volt (or kHz.)=C1*C0/65535
CL	(16852/53) Count at lower plot limit (=PL*CM)
CH	Count at upper plot limit =PH*CM
R	(16850/51) Plot ratio =(CH-CL)/(max. y val.)
	Y-position given by (count-CL)/R
PL	value at bottom of graph
PH	value at top of graph
DP	order of magnitude of accuracy (for # of decimal places)

FL	(16844/45) Which file? (screen)
N	(16849) Number of screens
D	(16846/47) Start address of data
VL	Voltage to print in left column
VH	Voltage to print in right column

Routines

FL	line 7900. Input which file (into FL)
SS	line 7950. Set "number of screens" into N.
FS	line 8200. Set C0 values for files being acquired.
FR	line 8300. Read C0 for that file and evaluate CM, TB, SR, CL, and CH.
C0	line 8400. C0py or Continue
D\$	Data (2048 points total
Y\$,Z\$	Keyboard input.

Enter the listing below, then RUN it. When it stops, delete lines 10 and 11, replacing them with the "Simple Loader Program" given. RUN this program and input the numbers shown in the "Decimal Listing" table. Start with the top left column and work horizontally. Ex: 169, ENTER, 166, ENTER, 185, ENTER, 166, ENTER and so forth. When the screen fills, press CONT and ENTER to continue. When finished, delete the loader program and enter the Basic listing.

```

1 PRINT 0+0+0+0+0+0+0+0+0+0+0
+0+0+0+0+0+0+0+0+0+0+0+0+0
+0+0+0+0+0+0+0+0+0+0+0+0+0
+0+0+0+0+0+0+0+0+0+0+0+0+0
4 POKE 16394,8
5 POKE 16395,0
6 POKE 16419,8
7 POKE 16420,0
8 REM DEPT-0000 ACQUISITION 1
9 REM BY F.NACHBAUR 1984
10 POKE 16513,234
11 REM RUN, THEN DELETE 10,11.
    START M/C LOADER HERE,
    DELETE WHEN M/C ENTERED,

```

```
10 FOR X=16514 TO 16843
20 PRINT X;" ";
30 INPUT Y
40 POKE X,Y
50 PRINT Y
60 NEXT X
```

[illegible]

```

8 REM W/F DATA ACQUISITION
9 REM BY F.NACHBAUR 1984
10 GOTO 5000
12 REM
15 REM GOTO 500 TO RESTART
500 CLS
510 POKE BL,N0
520 SLOW
530 PRINT TAB 6;"W/F DATA ACQ"
540 PRINT TAB 6;"PROGRAM 1, V 1."
550 TAB 6;"SAMPLE RATE 3-1000 MSE"
560 C,,L$;TAB 3;"(C)1984","BY F.N"
570 ACHBAUR,L$
580 PRINT "CHOOSE AN OPTION"
590 "1. ACQUIRE DATA (SINGLE SCRE"
600 EN)"2. PLOT DATA","3. PRINT D"
610 ATA",L$;"4. RE-SET PLOT LIMITS"
620 "5. INITIALIZE",L$;"6. MULTI-S"
630 CREEN ACQUISITION","7. SIDE-WIND"
640 ER PLOT (TS2040)",L$,"A. STOP"
650 "8. SAVE"
660 IF INKEY$="" THEN GOTO 550
670 LET Z$=INKEY$

```

69

```

1160 FAST
1170 RAND USR AQ
1180 SLOW
1190 PRINT AT 21,N0;"==DATA==";
1200 GOSUB CO
1210 GOTO H5
1220 PRINT AT 21,N0;"ANALYSIS TO";
1230 IF INKEY$("<")="" THEN GOTO 1230
1240 IF INKEY$="" THEN GOTO 1240
1250 GOTO 1160
1260 PRINT AT 21,N0;"CHECKING..."

1270 LET C=USR 16536
1280 IF (C AND Z$="N") OR (NOT C AND Z$="Z") THEN GOTO 1160
1290 GOTO 1270
2000 REM 2000
2010 CLS
2020 GOSUB FI
2030 GOSUB FR
2040 LET R=INT ((CH-CL)/40)
2050 CLS
2060 PRINT (" "+STR$ PH+" V ----"
-----" ) ( TO 32
)

2070 RAND USR 16536
2075 POKE BL,N0
2080 PRINT AT 00,N0;"*";AT 21,N0
;PL;TAB 00;"<---TIME--->";TAB 27
;TB;TAB 14;"---";FL
2085 POKE BL,N2
2090 RAND R
2100 POKE 16850,PEEK 51
2110 POKE 16851,PEEK 52
2120 POKE 16853,INT (CL/TF5)
2130 POKE 16852,CL-TF5*PEEK 1685
3

2140 RAND USR 16776
2150 GOSUB CO
2160 GOTO H5
3000 REM 3000
3010 GOSUB FI
3020 FAST
3030 GOSUB FR
3040 CLS
3050 PRINT "FILE NO: ",FL,"TIME
BASE = ",TB," SEC.", "SAMPLE RA
TE = ",SR," MSEC",L$," SEC"
3060 LET DP=00*INT (.2+LN CH/LN
00)
3070 FOR A=N1 TO 32
3080 FAST
3090 LET D=N2*(A-N1)+PEEK 16846+
TF5*PEEK 16847
3100 LET VL=(PEEK D+TF5*PEEK (D+
1))/CH
3110 LET VL=(INT (VL*DP))/DP
3120 LET VH=(PEEK (D+64)+TF5*PEE
K (D+65))/CH
3130 LET VH=(INT (VH*DP))/DP
3140 PRINT A;TAB 3;INT (A*SR+N1/
N2);TAB 9;VL;TAB 16;" ";A+32;TAB
20;INT ((A+32)*SR+N1/N2);TAB 26
;VH
3150 IF A=14 THEN GOSUB CO
3160 IF A=14 THEN CLS
3170 NEXT A
3180 GOSUB CO
3190 GOTO H5
4000 PRINT "PRESENT LIMITS: ", "M
IN. VOLTAGE=";PL,"MAX. VOLTAGE="
;PH
4010 PRINT "CHANGE TO: ", "MIN.
="
4020 INPUT PL
4030 PRINT PL,"MAX.=";
4040 INPUT PH
4050 PRINT PH,"OK?"
4060 IF INKEY$="" THEN GOTO 4060
4070 IF INKEY$="Y" THEN GOTO H5
4080 CLS
4090 GOTO 4000
5000 PRINT "END OF DATA"
5010 DIM D$(4096)
5020 LET L$=""

5030 LET N0=0
5040 LET N1=1
5050 LET N2=2
5060 LET 00=10
5070 LET TF5=256
5080 LET S1=16434
5090 LET S2=16435
5100 LET H5=500
5110 LET BL=16418
5120 LET FI=7900
5130 LET SS=7950
5140 LET FS=8200
5150 LET FR=8300
5160 LET CO=8400

```

```

5170 LET AQ=16594
5180 LET PL=N0
5190 LET PH=N1
5200 LET N=N1
5210 PRINT "SET SW. 3,6,7; MEASU
RE VREF",,"VREF=";
5220 INPUT VREF
5230 PRINT VREF
5240 POKE 16540,255
5250 POKE 16541,255
5260 FAST
5270 LET C=USR 16536
5280 SLOW
5290 PRINT "MAX COUNT=";C
5300 LET C1=C/VREF
5310 PRINT "IV COUNT AT MAX TIME
=";C1;
5320 PRINT "SET TIME-BASE; PRE
SS ENTER WHEN READY, NOTE TIM
E WHEN DISPLAY RE-APPEARS."
5330 IF INKEY$("<")CHR$ 118 THEN GO
TO 5330
5340 FAST
5350 LET FL=N1
5360 GOSUB 7915
5370 LET CO=65535
5380 GOSUB FS
5390 PRINT "LAST COUNT=";USR 165
66
5400 SLOW
5410 PRINT "TIME=";
5420 INPUT TREF
5430 PRINT TREF
5440 GOSUB CO
5450 GOTO H5
6000 REM 6000
6010 CLS
6020 PRINT "START AT "
6030 GOSUB FI
6040 PRINT
6050 GOSUB SS
6055 IF FL+N>33 THEN GOTO 6000
6060 PRINT " (EACH SCREEN) ="
6070 GOTO 1030
7000 REM 7000
7010 GOSUB FI
7020 GOSUB SS
7025 IF FL+N>33 THEN GOTO 7000
7030 GOSUB FR
7040 LET R=((CH-CL)/60)
7050 LET DR=N0
7060 PRINT "DRY RUN?"
7070 IF INKEY$("<") THEN GOTO 707
0
7080 IF INKEY$="" THEN GOTO 7080
7090 IF INKEY$="Y" THEN LET DR=N
1
7100 FAST
7110 IF NOT DR THEN LPRINT "TB
=";TB," SEC PER SCREEN", "SAMPLE
RATE=";SR," MSEC", "TOTAL TIME="
;N*TB," SEC"
7120 IF NOT DR THEN LPRINT "STR
$ PL;"=MIN. VOLTS";TAB 21;"M
AX=";PH
7130 LET D=PEEK 16846+TF5*PEEK 1
6847
7140 LET F=FL
7150 LET C=N0
7160 LET L=N0
7170 LET TM=N0
7180 FOR L=L TO L+21
7190 SCROLL
7200 IF L>=32*N THEN GOTO 7340
7210 PRINT AT 21,30;" ";CHR$ (00
+NOT TM);AT 21,N0;"T";AT 21,N0;(
CHR$ (F+156)+-----+-----
+-----I") AND NOT C
7220 FOR Y=N1 TO N0 STEP -N1
7230 LET B=PEEK D+TF5*PEEK (D+N1
)
7240 LET X=INT ((B-CL)/R+N1/N2)
7250 IF X<64 AND X>=N0 THEN PLOT
X,Y
7260 LET D=D+N2
7270 NEXT Y
7280 LET TM=TM+N1
7290 LET C=C+N1
7300 IF TM=5 THEN LET TM=N0
7310 IF C<>32 THEN GOTO 7340
7320 LET C=N0
7330 LET F=F+N1
7340 NEXT L
7350 IF NOT DR THEN COPY
7360 IF DR THEN PAUSE 4E4
7370 IF L<32*N THEN GOTO 7180
7380 GOTO H5
7900 LET N$="WHICH SCREEN? (01-3
2) = "
7905 GOSUB 8000
7910 LET FL=A
7915 POKE 16844,FL
7920 POKE 16845,N0
7925 RAND USR 16611

```



```

7930 RETURN
7950 LET N$="HOW MANY SCREENS? ("
  TO "+STR$(33-FL)+")":
7960 GOSUB 8000
7970 LET N=A
7980 RETURN
8000 PRINT ,N$;
8010 LET Z$=INKEY$
8020 IF Z$<"0" OR Z$>"9" THEN GO
  TO 8010
8030 PRINT Z$;
8040 IF INKEY$<>" " THEN GOTO 804
  0
8050 LET Y$=INKEY$
8060 IF Y$=" " THEN GOTO 8050
8070 IF CODE Y$=118 THEN GOTO 81
  10
8080 PRINT Y$;
8090 IF Y$<"0" OR Z$>"3" OR (Z$=
  "3" AND Y$="2") OR Y$>"9" THEN G
  OTO 8160
8100 GOTO 8130
8110 LET Y$=Z$
8120 LET Z$="0"
8130 PRINT " "
8140 LET A=00*VAL Z$+VAL Y$
8150 RETURN
8160 CLS
8170 GOTO 8200
8200 RAND 00
8210 FOR F=FL TO FL+N-1
8220 POKE 16854+2*F,PEEK S1
8230 POKE 16855+2*F,PEEK S2
8240 NEXT F
8250 RETURN
8300 LET C0=PEEK (16854+N2*FL)+T
  F5*PEEK (16855+N2*FL)
8310 LET CH=C1+C0/65535
8320 LET TB=INT (TREF*C0/655.35+
  .5)/00/00
8330 LET SR=N2*H5*TB/64
8340 LET CL=PL+CM
8350 LET CH=PH+CM
8360 RETURN
8400 POKE 16418,N0
8410 SLOW
8420 PRINT AT 23,N0;"- COPY ";T
  AB 23;"- CONT "
8430 POKE 16418,N2
8440 IF INKEY$="Z" THEN COPY
8450 IF INKEY$="C" THEN RETURN
8460 GOTO 8440
9970 CLS
9974 PRINT "
9976 IF INKEY$<>" " THEN GOTO 997
  6
9978 IF INKEY$=" " THEN GOTO 9978
9980 CLS
9982 SAVE "VDAQ"
9984 GOTO H5
9990 CLS
9991 POKE BL,0
9992 PRINT AT 21,0;" GOTO 500 T
  O RE-START","ENTER" TO LIST "
9993 FAST
9999 POKE 16384,89

```

HEX-ASSEMBLY LISTING (Annotated as best I can)

ADDR	HEX	NAME	DEC	ADDR	CHR\$
40800	A9	169	168514		
40801	A8	168	168515		
40802	A9	169	168516		
40803	A8	168	168517		
40804	80	128	168518		
40805	A8	168	168519		
40806	A8	168	168520		
40807	A8	168	168521		
40808	A8	168	168522		
40809	A8	168	168523		
4080A	B3	179	168524		
4080B	8D	157	168525		
4080C	A5	165	168526		
4080D	A4	164	168527		
4080E	A0	160	168528		
4080F	00	0	168529		
40810	00	0	168530		
40811	00	0	168531		
40812	1D	29	168532		
40813	1B	27	168533		
40814	1F	31	168534		
40815	00	0	168535		
40816	00	0	168536		

ADDR	HEX	CODE	MNEMONIC	ACC, A POINT
40817	00			
40818	00			
40819	00			
4081A	00			
4081B	00			
4081C	00			
4081D	00			
4081E	00			
4081F	00			
40820	00			
40821	00			
40822	00			
40823	00			
40824	00			
40825	00			
40826	00			
40827	00			
40828	00			
40829	00			
4082A	00			
4082B	00			
4082C	00			
4082D	00			
4082E	00			
4082F	00			
40830	00			
40831	00			
40832	00			
40833	00			
40834	00			
40835	00			
40836	00			
40837	00			
40838	00			
40839	00			
4083A	00			
4083B	00			
4083C	00			
4083D	00			
4083E	00			
4083F	00			
40840	00			
40841	00			
40842	00			
40843	00			
40844	00			
40845	00			
40846	00			
40847	00			
40848	00			
40849	00			
4084A	00			
4084B	00			
4084C	00			
4084D	00			
4084E	00			
4084F	00			
40850	00			
40851	00			
40852	00			
40853	00			
40854	00			
40855	00			
40856	00			
40857	00			
40858	00			
40859	00			
4085A	00			
4085B	00			
4085C	00			
4085D	00			
4085E	00			
4085F	00			
40860	00			
40861	00			
40862	00			
40863	00			
40864	00			
40865	00			
40866	00			
40867	00			
40868	00			
40869	00			
4086A	00			
4086B	00			
4086C	00			
4086D	00			
4086E	00			
4086F	00			
40870	00			
40871	00			
40872	00			
40873	00			
40874	00			
40875	00			
40876	00			
40877	00			
40878	00			
40879	00			
4087A	00			
4087B	00			
4087C	00			
4087D	00			
4087E	00			
4087F	00			
40880	00			
40881	00			
40882	00			
40883	00			
40884	00			
40885	00			
40886	00			
40887	00			
40888	00			
40889	00			
4088A	00			
4088B	00			
4088C	00			
4088D	00			
4088E	00			
4088F	00			
40890	00			
40891	00			
40892	00			
40893	00			
40894	00			
40895	00			
40896	00			
40897	00			
40898	00			
40899	00			
4089A	00			
4089B	00			
4089C	00			
4089D	00			
4089E	00			
4089F	00			
408A0	00			
408A1	00			
408A2	00			
408A3	00			
408A4	00			
408A5	00			
408A6	00			
408A7	00			
408A8	00			
408A9	00			
408AA	00			
408AB	00			
408AC	00			
408AD	00			
408AE	00			
408AF	00			
408B0	00			
408B1	00			
408B2	00			
408B3	00			
408B4	00			
408B5	00			
408B6	00			
408B7	00			
408B8	00			
408B9	00			
408BA	00			
408BB	00			
408BC	00			
408BD	00			
408BE	00			
408BF	00			
408C0	00			
408C1	00			
408C2	00			
408C3	00			
408C4	00			
408C5	00			
408C6	00			
408C7	00			
408C8	00			
408C9	00			
408CA	00			
408CB	00			
408CC	00			
408CD	00			
408CE	00			
408CF	00			
408D0	00			
408D1	00			
408D2	00			
408D3	00			
408D4	00			
408D5	00			
408D6	00			
408D7	00			
408D8	00			
408D9	00			
408DA	00			
408DB	00			
408DC	00			
408DD	00			
408DE	00			
408DF	00			
408E0	00			
408E1	00			
408E2	00			
408E3	00			
408E4	00			
408E5	00			
408E6	00			
408E7	00			
408E8	00			
408E9	00			
408EA	00			
408EB	00			
408EC	00			
408ED	00			
408EE	00			
408EF	00			
408F0	00			
408F1	00			
408F2	00			
408F3	00			
408F4	00			
408F5	00			
408F6	00			
408F7	00			
408F8	00			
408F9	00			
408FA	00			
408FB	00			
408FC	00			
408FD	00			
408FE	00			
408FF	00			
40900	00			
40901	00			
40902	00			
40903	00			
40904	00			
40905	00			
40906	00			
40907	00			
40908	00			
40909	00			
4090A	00			
4090B	00			
4090C	00			
4090D	00			
4090E	00			
4090F	00			
40910	00			
40911	00			
40912	00			
40913	00			
40914	00			
40915	00			
40916	00			
40917	00			
40918	00			
40919	00			
4091A	00			
4091B	00			
4091C	00			
4091D	00			
4091E	00			
4091F	00			
40920	00			
40921	00			
40922	00			
40923	00			
40924	00			
40925	00			
40926	00			
40927	00			
40928	00			
40929	00			
4092A	00			
4092B	00			
4092C	00			
4092D	00			
4092E	00			
4092F	00			
40930	00			
40931	00			
40932	00			
40933	00			
40934	00			
40935	00			
40936	00			
40937	00			
40938	00			
40939	00			
4093A	00			
4093B	00			
4093C	00			
4093D	00			
4093E	00			
4093F	00			
40940	00			
40941	00			
40942	00			

THE 16-BIT DIVISION ROUTINE
 BELOW IS PRINTED COURTESY OF
 BOB BEACH. CONSULT THE TEXT FOR
 HIS ZX COMPILER AND FOR OTHER
 ROUTINES AND ENTRY POINTS.

```

4117 0E01 LD C,01
4119 7C LD A,H
411A DA XOR D
411B E880 AND 80
411D 47 LD B,A
411E 08 PUSH DE
411F 0C06A41 CALL 416A
4120 0E03 EX (SP),HL
4123 0C06A41 CALL 416A
4126 0E03 EX DE,HL
4127 7F11 POP HL
4128 7F01 LD A,C
4129 0A77 AND A
412A 0A36 JR NZ 4162
412B 0A01 LD A,B
412C 0A01 OR C
412D 0E05 PUSH HL
412E 0E05 SBC HL,HL
412F 0E05 SBC HL,DE
4131 0E05 SBC HL,DE
4133 0E05 PUSH HL
4134 0E05 POP HL
4135 0E05 POP DE
4136 0E05 LD HL,0000
4137 0E05 SCF
4138 0E05 CCF
4139 0E05 PUSH AF
413A 0E05 LD A,D
413B 0E05 LD D,E
413C 0E05 LD E,08
413D 0E05 ADD HL,HL
413E 0E05 JR NC 4148
413F 0E05 ADD A,A
4140 0E05 ADD HL,BC
4141 0E05 JR 4151
4142 0E05 ADD A,A
4143 0E05 ADD HL,BC
4144 0E05 JR C 4180
4145 0E05 SBC HL,BC
4146 0E05 DEC A
4147 0E05 INC A
4148 0E05 DEC A
4149 0E05 INC A
414A 0E05 JR NZ 4140
414B 0E05 LD A,08
414C 0E05 LD A,08
414D 0E05 LD A,08
414E 0E05 LD A,08
414F 0E05 LD A,08
4150 0E05 LD A,08
4151 0E05 LD A,08
4152 0E05 LD A,08
4153 0E05 LD A,08
4154 0E05 LD A,08
4155 0E05 LD A,08
4156 0E05 LD A,08
4157 0E05 LD A,08
4158 0E05 LD A,08
4159 0E05 LD A,08
415A 0E05 LD A,08
415B 0E05 LD A,08
415C 0E05 LD A,08
415D 0E05 LD A,08
415E 0E05 LD A,08
415F 0E05 LD A,08
4160 0E05 LD A,08
4161 0E05 LD A,08
4162 0E05 LD A,08
4163 0E05 LD A,08
4164 0E05 LD A,08
4165 0E05 LD A,08
4166 0E05 LD A,08
4167 0E05 LD A,08
4168 0E05 LD A,08
4169 0E05 LD A,08
416A 0E05 LD A,08
416B 0E05 LD A,08
416C 0E05 LD A,08
416D 0E05 LD A,08
416E 0E05 LD A,08
416F 0E05 LD A,08
4170 0E05 LD A,08
4171 0E05 LD A,08
4172 0E05 LD A,08
4173 0E05 LD A,08
4174 0E05 LD A,08
4175 0E05 LD A,08
4176 0E05 LD A,08
4177 0E05 LD A,08
4178 0E05 LD A,08
4179 0E05 LD A,08
417A 0E05 LD A,08
417B 0E05 LD A,08
417C 0E05 LD A,08
417D 0E05 LD A,08
417E 0E05 LD A,08
417F 0E05 LD A,08
4180 0E05 LD A,08
4181 0E05 LD A,08
4182 0E05 LD A,08
4183 0E05 LD A,08
4184 0E05 LD A,08
4185 0E05 LD A,08
4186 0E05 LD A,08
4187 0E05 LD A,08
4188 0E05 LD A,08
4189 0E05 LD A,08
418A 0E05 LD A,08
418B 0E05 LD A,08
418C 0E05 LD A,08
418D 0E05 LD A,08
418E 0E05 LD A,08
418F 0E05 LD A,08
4190 0E05 LD A,08
4191 0E05 LD A,08
4192 0E05 LD A,08
4193 0E05 LD A,08
4194 0E05 LD A,08
4195 0E05 LD A,08
4196 0E05 LD A,08
4197 0E05 LD A,08
4198 0E05 LD A,08
4199 0E05 LD A,08
419A 0E05 LD A,08
419B 0E05 LD A,08
419C 0E05 LD A,08
419D 0E05 LD A,08
419E 0E05 LD A,08
419F 0E05 LD A,08
41A0 0E05 LD A,08
41A1 0E05 LD A,08
41A2 0E05 LD A,08
41A3 0E05 LD A,08
41A4 0E05 LD A,08
41A5 0E05 LD A,08
41A6 0E05 LD A,08
41A7 0E05 LD A,08
41A8 0E05 LD A,08
41A9 0E05 LD A,08
41AA 0E05 LD A,08
41AB 0E05 LD A,08
41AC 0E05 LD A,08
41AD 0E05 LD A,08
41AE 0E05 LD A,08
41AF 0E05 LD A,08
41B0 0E05 LD A,08
41B1 0E05 LD A,08
41B2 0E05 LD A,08
41B3 0E05 LD A,08
41B4 0E05 LD A,08
41B5 0E05 LD A,08
41B6 0E05 LD A,08
41B7 0E05 LD A,08
41B8 0E05 LD A,08
41B9 0E05 LD A,08
41BA 0E05 LD A,08
41BB 0E05 LD A,08
41BC 0E05 LD A,08
41BD 0E05 LD A,08
41BE 0E05 LD A,08
41BF 0E05 LD A,08
41C0 0E05 LD A,08
41C1 0E05 LD A,08
41C2 0E05 LD A,08
41C3 0E05 LD A,08
41C4 0E05 LD A,08
41C5 0E05 LD A,08
41C6 0E05 LD A,08
41C7 0E05 LD A,08
41C8 0E05 LD A,08
41C9 0E05 LD A,08
41CA 0E05 LD A,08
41CB 0E05 LD A,08
41CC 0E05 LD A,08
41CD 0E05 LD A,08
41CE 0E05 LD A,08
41CF 0E05 LD A,08
41D0 0E05 LD A,08
41D1 0E05 LD A,08
41D2 0E05 LD A,08
41D3 0E05 LD A,08
41D4 0E05 LD A,08
41D5 0E05 LD A,08
41D6 0E05 LD A,08
41D7 0E05 LD A,08
41D8 0E05 LD A,08
41D9 0E05 LD A,08
41DA 0E05 LD A,08
41DB 0E05 LD A,08
41DC 0E05 LD A,08
41DD 0E05 LD A,08
41DE 0E05 LD A,08
41DF 0E05 LD A,08
41E0 0E05 LD A,08
41E1 0E05 LD A,08
41E2 0E05 LD A,08
41E3 0E05 LD A,08
41E4 0E05 LD A,08
41E5 0E05 LD A,08
41E6 0E05 LD A,08
41E7 0E05 LD A,08
41E8 0E05 LD A,08
41E9 0E05 LD A,08
41EA 0E05 LD A,08
41EB 0E05 LD A,08
41EC 0E05 LD A,08
41ED 0E05 LD A,08
41EE 0E05 LD A,08
41EF 0E05 LD A,08
41F0 0E05 LD A,08
41F1 0E05 LD A,08
41F2 0E05 LD A,08
41F3 0E05 LD A,08
41F4 0E05 LD A,08
41F5 0E05 LD A,08
41F6 0E05 LD A,08
41F7 0E05 LD A,08
41F8 0E05 LD A,08
41F9 0E05 LD A,08
41FA 0E05 LD A,08
41FB 0E05 LD A,08
41FC 0E05 LD A,08
41FD 0E05 LD A,08
41FE 0E05 LD A,08
41FF 0E05 LD A,08

```

PLOT A SCREEN

```

4188 0E00 LD A,00 : INIT. X COUNTER
4189 320041 LD (41D0),A : STORE COUNTER
418A 0E00 LD HL,(41CE) : GET START ADDRESS
418B 0E00 LD D,00 : PUT COUNTER IN D
418C 0E00 LD E,A : AND E
418D 0E00 ADD HL,DE : AND ADD TO ADDRESS
418E 0E00 ADD HL,DE : TWICE
418F 0E00 LD C,(HL) : GET DATA INTO BC
4190 0E00 INC HL
4191 0E00 LD B,(HL)
4192 0E00 PUSH BC
4193 0E00 POP HL
4194 0E00 LD DE,(41D4) : NOW IT'S IN HL
4195 0E00 SBC HL,DE : GET CL INTO DE
4196 0E00 JR C 41C1 : HL=COUNT-CL
4197 0E00 LD DE,(41D2) : GET R INTO DE
4198 0E00 CALL 4117 : DIVIDE HL/DE
4199 0E00 NOP : RESULT IN HL
419A 0E00 NOP : NOPS AND INC HL'S
419B 0E00 INC HL : USED TO TRIM
419C 0E00 INC HL : VERTICAL OFFSET
419D 0E00 LD A,H : CHECK HI-BYTE
419E 0E00 CP 00 : FOR OVERRANGE
419F 0E00 JR C 41C1 : SKIP IF TOO HIGH
41A0 0E00 LD A,L : CHECK LO-BYTE
41A1 0E00 CP 20 : FOR OVERRANGE
41A2 0E00 JR NC 41C1 : SKIP IF TOO HIGH
41A3 0E00 LD B,L : Y COORD. IN B
41A4 0E00 LD A,(41D0) : GET X COORD.
41A5 0E00 LD C,A : AND PUT INTO C
41A6 0E00 LD A,80 : SET UP FOR -PLOT-
41A7 0E00 CALL 05B2 : CALL -PLOT-
41A8 0E00 LD A,(41D0) : GET COUNTER
41A9 0E00 INC A : AND INCR. IT
41AA 0E00 CP 40 : ALL DONE?
41AB 0E00 RET Z : YES, GO HOME
41AC 0E00 JR 4189 : NO, LOOP BACK
41AD 0E00 NOP

```

With this issue we start a column strictly for short tips, hints, and curiosities submitted by our readers, and items we come up with that don't fit anywhere else.

Listing Scanner

by Paul Holmgren

Here is a short listing you might find useful. I developed it after trying to find the mistakes, and there were several, in a 12K long program that I hand-loaded from a magazine listing. My eyes could not take the strain of staring at the TV and then the listing, back and forth over and over. Ergo, this routine, which can be placed anywhere in a listing. It uses about 200 bytes, and works great in FAST mode. I don't suggest using VAL within the loop because of its effect on program speed.

```
10 LET D=VAL "16509"
15 CLS
20 LET G=PEEK D*256+PEEK (D+SG
N PI)
25 LET D=D+VAL "4"
30 PRINT AT PI,4-LEN STR$ G;G;
35 IF PEEK D=126 THEN LET D=D+
6
40 IF PEEK D=118 THEN LET G=NO
T G
45 IF G THEN PRINT CHR$ PEEK D
;
50 LET D=D+SGN PI
55 IF G THEN GOTO 35
60 IF PEEK D=118 THEN STOP
65 PAUSE D
70 GOTO VAL "15"
```

Hot Z-II Fix

by Fred Nachbaur

The 16K Hot-Z II has a minor "bug" which causes the program to wipe the bytes from 8000h-8020h when you QUIT the program. If you're storing complete BASIC programs in the 8009-BFFF range, this will "eat" some system variables stored up there.

The author supplies this fix: load HOT-Z, and then use it to load a "slave" copy of itself at C009 as described in the documentation. At CDDF there will be a LD BC,009F; change this to LD BC,007F, and save the "slave" to a new tape; when using this in the future, the bytes 8000-8020 will be left alone. When HZII boots itself, the address of this command becomes 581B. You might also want to change the version number in the "cover" to reflect the change, e.g. add a "-1."

Running On 64K

Here's that article, promised in issue 1:3, to help get new 64K RAM owners off the ground.

Since the Z80 has 16 address lines, the number of possible address combinations is $2^{16}=65536$ (or 64K, or 10000h; one greater than FFFFh, the highest address). If we look at the possible states of A14 and A15 (the most significant bits) the memory address range breaks down into four 16K "blocks." If we also use A13, we have eight 8K blocks, and so on. But let's just look at each 16K block and what each is used for in turn. Think of your RAM as a building with a basement and mezzanine, a main floor, and first and second stories.

The bottom 16K is split into the "basement" (0-8K = ROM) and the "mezzanine" (8-16K = user transparent RAM). The "basement" is of course not usable for your programming, but some RAMs decode this area to 8-16K to give you an "alternate mezzanine." The "main floor" is where your entire BASIC system is normally at home. The two stories (32-48 and 48-64K) are much like the main floor, but there are restrictions. Before we look at these though, here are a few words on the ZX display.

The ZX81/ TS1000-1500 is an "interrupt-driven" machine, using the interrupts to switch between actual computing and generating the TV display. When the display is being created, the "second story" (48-64K) of addresses is used. Presumably to make the decoding simple, only the A15 bit is used internally, so the entire 32-64K range is off-limits to running machine-code. On many packs you can add the additional "Oliger mod" decoding to let you run machine-code in the "first floor", but 48-64K is still tied up by the system hardware. This won't work on all RAMs; it works with JLO's (of course) and others, but NOT on Byte-Back's UM64, nor on some "strange" packs like my little JRS 64.

The "mezzanine" is perhaps the nicest part of the house; you can run machine-code here, and it's a great place to keep MC utilities. The party always congregates here, though, so it gets a little crowded; therefore you'll have to plan it out before you start filling it.

You usually will run BASIC only in the "main floor," but can push your BASIC instruction file up into the higher stories to allow up to 48K programs. Problems occur when part of the display file (riding on top of the program) is on the main floor and part on the first story; avoid this straddling during program entry by moving the DFILE up beyond the 32K mark in one step, in FAST mode, by entering or generating a huge REM statement (about 1K long) when you're pushing the limit. After you've added enough of your own program, you can delete the dummy REM. A better way than typing REM xxxx (1000 times) is Memotech's suggestion to enter PRINT 0+0+0.. (125 times) and make use of that extravagant floating-point number storage. You can also run BASIC in the mezzanine or top stories by jimmying the "Next Line"

system variable at 16425, but it's a pain; and you can't use GOTO/SUB inside such areas. (Thanks to Ray Kingsley for that one).

My favorite usage of the top stories is for storage of entire 16K programs; you can use block transfer routines located in the mezzanine to boot the whole 16K-32K range, SV's and all, into whichever block, and later another similar routine zaps it back down about as fast as you can type RAND USR 8192. This is particularly handy when working with the 16K HOT-Z II. (I usually use it with BIG REM, useful for working with machine code written for the low-end of the BASIC area, but it's not vital.) I then load the program being worked on in the 32-48K range, and use 48K-64K to store HOT Z, BIG REM and all, when it's not being used (as when testing the program after changing and SAVING back to tape, quitting HOT Z, and loading back the modified program).

A quick note about RAMTOP... 64K RAMs generally come with documentation that tells you to POKE it to 255/255. Well, you don't have to do that unless you need a full 64K. I usually leave it at the 16K default value by omitting the POKEs; the machine now thinks it's only 16K, but 32-64K is still available to PEEK/POKE or otherwise transfer (LDIR/LDDR) data. You might want to POKE it to 24K (0/160) to allow two 24K blocks, one for running and one for storage of a 24K program, or to 32K (0/192) so you can LOAD a 32K program while having a 16K program "in the wings." Anything above RAMTOP is immune to NEW (and reset, provided ramtop is at or above 32767); so normally it will still be there even if you crash. (The exception is if you've modified your ROM to initialize all the way to 64K on reset; which is why I left mine as-is). Same holds true for the mezzanine (8-16K), regardless of whether or not the ROM initialization has been changed.

Change RAMTOP Without NEW

By Fred Nachbaur

You may have wanted to change RAMTOP without wiping out what you have in memory. It is not enough to simply POKE 16388 and 16389 with the desired RAMTOP, since the machine-code stack stays in the same place until you relocate them using NEW. Several "fixes" have been reported, including one using a ROM call in SWN 1:3; an observant reader pointed out that it simply didn't work. Since then, I've been looking into how the various programs that accomplish this go about it, and found there are several different approaches. My favorite is used in the initialization routine for G. Russell's "HRP" program. In its simplest form, the routine goes like this:

```

215B40 LD HL,TADD
F9 LD SP,HL
21nnnn LD HL, desired RAMTOP
220440 LD (RMTP),HL
2B DEC HL
363E LD (HL),3E
2B DEC HL
F9 LD SP,HL
2B DEC HL
2B DEC HL
220240 LD (ERSP),HL
C37506 JP 0675

```

Notice how the routine uses system variable T-ADDR, which the manual claims is "very unlikely to be useful." (Heh, heh...) Also note the lack of a RET at the end; instead it jumps into the ROM (at an "uncharted" entry point, no less) from whence the ROM returns you to BASIC. The charm of this one is that you can use it within a program and keep right on running on return. Don't call it from within a GOSUB or m/c CALL, and don't PUSH anything you want to keep, as the stack is reset to "empty."

Algebra

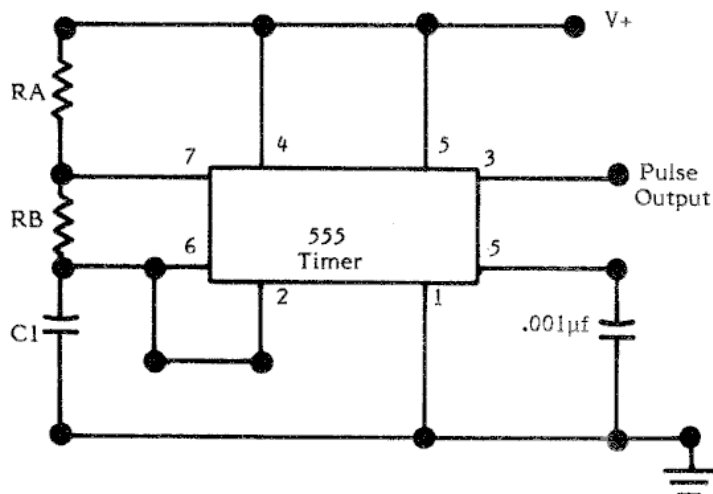
By Bill Yotter
Soft-Way
3308 Midway Dr., Dept. 124
San Diego, CA 92110

Here's a neat program that Bill has allowed us to print, hoping that some of you will avail yourselves of his other programs. Drop him a line at the above address for his current product list.

This program, like all Soft-way programs, is tightly coded and numbered consecutively. You may wish to re-number using a toolkit (or simply entering each line number * 10) if like me you prefer more elbow room. (No computed GOTO/GOSUBs so toolkits work fine). It allows you to enter up to 20 formulas, including transcendental functions (LN, SIN, etc.) and bracketing. The dependent variable (on the left) is two characters, the independent variables must be one character. Work with any formula, then DISPLAY updates each before showing the values of the dependents. (You can't put dependent variables over to the right side directly, e.g. in the example you can't use $FQ = 1/PR$; see what you can come up with to do this if you're interested. It's trickier than it looks!)

The best way to learn to use ALGEBRA is to follow Bill's example below. It solves the 555 timer IC in the "astable multi" mode (pulse generator). Pretty nifty, no?

555 Astable Multivibrator Signal Source



To enter line 0: enter as line 1, then POKE 16510,0.

```

0 REM ALGEBRA (C) SOFTWAY 83
1 GOTO 12
2 DIM A$(20,29)
3 LET C$=""
4 CLS
5 PRINT TAB 5;"MAKE DESIRED C
HANGES:"
6 FOR X=1 TO 20
7 PRINT AT X+1,0;X;" ";A$(X)
8 INPUT B$
9 IF B$<>" " THEN LET A$(X)=B$
10 PRINT AT X+1,0;X;" ";A$(X)
11 NEXT X
12 CLS
13 FOR X=1 TO 20
14 PRINT AT X,0;X;" ";A$(X)
15 NEXT X
16 PRINT AT 21,2;"SELECT DESIR
ED EQUATION:"
17 INPUT X
18 CLS
19 PRINT AT 4,2;A$(X)
20 FOR Y=4 TO 29
21 IF A$(X,Y)>="A" AND A$(X,Y)
<="Z" THEN GOSUB 32
22 NEXT Y
23 IF A$(X,3)="" THEN PRINT
TAB 5;A$(X, TO 2);" = ";VAL
A$(X,4 TO )
24 PRINT AT 21,0;"DISPLAY REPE
AT LIST CHANGE SAVE"

```

EXAMPLE: DISPLAY - REPEAT

@ = "Enter", # = "space."

```

RUN 2 @ @ @
####A#B#C @ @ @
CT= .693*(A+B)*C @ @ @
DT= .693*B*C @ @ @
PR= .693*(A+2*B)*C @ @ @
FQ= 1.44/((A+2*B)*C) @ @ @
DC= B/(A+2*B) @ @ @ @

```

where

CT= Charge Time
DT= Discharge Time
PR= Period (= CT+DT)
FQ= Frequency
DC= Duty Cycle

```

25 PAUSE 1000
26 IF INKEY$="D" THEN GOTO 49
27 IF INKEY$="S" THEN GOTO 48
28 IF INKEY$="R" THEN GOTO 18
29 IF INKEY$="L" THEN GOTO 12
30 IF INKEY$="C" THEN GOTO 3
31 GOTO 24
32 PRINT TAB 7;A$(X,Y);" = ?
33 INPUT B$
34 GOTO 44
35 IF B$="" THEN GOTO 33
36 POKE (PEEK 16425+256*PEEK 1
6426+5),CODE A$(X,Y)
37 LET C=VAL B$
38 PRINT VAL B$
39 LET C$=C$+A$(X,Y)
40 RETURN
41 POKE (PEEK 16425+256*PEEK 1
6426+5),CODE C$(0)
42 PRINT A
43 RETURN
44 FOR O=1 TO LEN C$
45 IF C$(O)=A$(X,Y) AND B$=""
THEN GOTO 41
46 NEXT O
47 GOTO 35
48 SAVE "ALGEBRA"
49 CLS
50 FOR O=1 TO 20
51 IF A$(O,3)="" THEN PRINT A
$(O, TO 3);" ";VAL A$(O,4 TO 29)
52 NEXT O
53 GOTO 24

```

: Independent variables -

A = RA
B = RB
C = C1

SELECT #3: Enter values for A, B, C.
PRESS D: Solutions for all formulas displayed.
PRESS R: Enter new value for A, @ @
PRESS D: All solutions updated for new A

Any set(s) of related equations can be used in this manner. Assignments are passed to all other formulas.

NOTE: You MUST set values to ALL variables prior to using the display routine. Also, XX/0 is not allowed.



Polar-Rectangular Conversion

This program converts between polar notation (magnitude and angle) and rectangular (or "Cartesian") notation ("X" and "Y" values) of vector quantities and complex numbers. Many scientific calculators have this function, which often proves useful if you deal with vectors or phasors. On the ZX81, this can be accomplished quite easily using the SIN, COS, ArcTan, and SQR functions. The program will fit into un-expanded ZX81/TS1000's.

PLR-CART goes a step beyond most calculators in allowing negative values for the magnitude in polar expressions. For example, "R=-5, A=30 deg." and "R=5, A=210 deg." are both mathematically correct, and indicate the same vector, but calculators generally don't accept expressions like the first, whereas this program will. Going the other way, from Cartesian to Polar, always gives a positive magnitude (generally considered preferable). The ZX81 trig functions automatically handle negative angles and angles greater than 2*PI radians (360 deg.), so we don't need to bother with this in the program.

To use the program, enter RUN with the machine in SLOW mode. You'll be prompted which way you want to convert, and then be asked to INPUT the appropriate variables. Going from Polar to Rectangular (Cartesian) you have the option of expressing the angle in degrees or radians. Pressing any key except BREAK after the answer appears gets you back to the beginning of the program.

The expression NOT PI is used throughout the program instead of zero to conserve memory space. (It takes six extra bytes to store the binary representation of a number in the program area.) Try it yourself - enter PRINT NOT PI. Review the manual section on logical operators if you're not sure why this works.

```

5 CLS
10 SLOW
20 PRINT "PLR-> CART "C";", "
CART-> PLR "P";"
30 LET A$=INKEY$
70 GOTO 170*(A$="C")+270*(A$="
P")+30
200 CLS
205 PRINT "DEG "D";", "RAD "R
...
210 LET A$=INKEY$
220 LET K=(A$="R")+(A$="D")*PI/
180
230 IF NOT K THEN GOTO 210
240 CLS
250 PRINT "R=";
255 INPUT R
260 PRINT R,"A=";
270 INPUT A
280 PRINT A;" ";A$;
285 LET X=R*COS (K*A)
287 LET Y=R*SIN (K*A)
290 PRINT "X=";X,"Y=";Y
295 IF INKEY$="" THEN GOTO 295
297 GOTO NOT PI
300 CLS
310 PRINT "X=";
320 INPUT X
330 PRINT X,"Y=";
340 INPUT Y
350 PRINT Y;
360 LET R=SQR (X*X+Y*Y)
370 LET A=ATN (Y/X)
380 IF X<NOT PI THEN LET A=A+PI
390 PRINT "R=";R,"A=";A;" R";"
=";A*180/PI;" D"
400 GOTO 295
410 CLEAR
420 SAVE "PLR-CART"
430 RUN

```

Straight Line Subroutine

This subroutine draws an unbroken line segment between pixel (X1,Y1) and (X2,Y2). It is based on the equation for a straight line:

$$Y = M * X + B$$

where M is the slope and B is the Y-intercept.

First the slope is evaluated in line 1020, after making sure that (X2-X1) is non-zero (division by zero is a no-no). If the slope is less than one, the line is drawn by incrementing X, otherwise the line is drawn by incrementing Y, resulting in an unbroken line in either case.

```

100 INPUT X1
110 INPUT Y1
120 INPUT X2
130 INPUT Y2
140 GOSUB 1000
150 RUN
1000 IF X2-X1=NOT PI THEN LET X2
=X2+.1
1005 LET M=(Y2-Y1)/(X2-X1)
1010 IF ABS M>1 THEN GOTO 1100
1020 LET I=Y1-M*X1
1030 FOR B=X1 TO X2 STEP SGN (X2
-X1)
1040 LET C=INT (M*B+I+.5)
1050 PLOT B,C
1060 NEXT B
1070 RETURN
1100 FOR C=Y1 TO Y2 STEP SGN (Y2
-Y1)
1110 LET B=(C+M*X1-Y1)/M
1115 PLOT B,C
1120 NEXT C
1130 RETURN

```

Enter RUN, then the values of X1, Y1, X2, and Y2. The program draws lines until you fill the screen too far and run out of memory.

If you have a 2K machine (or greater) you can use the subroutine to graph the vectors obtained in the PLR-CART program. Enter or LOAD PLR-CART, and add the subroutine. Now enter the following lines:

```

500 IF INKEY$="G" THEN GOTO 500
505 CLS
510 FOR B=1 TO 31
515 PRINT AT 11,B;" "
520 IF B<22 THEN PRINT AT B,15;
530 NEXT B
540 PRINT AT 0,0;"2";TAB 15;"Y
";TAB 31;"1";AT 11,0;"-";TAB 30;
"+X";AT 21,0;"3";TAB 15;"-";TAB
31;"4"
550 LET X1=31
560 LET X2=X+X1
570 LET Y1=20
580 LET Y2=Y+Y1
590 GOSUB 1000
600 GOTO 295

```

After results have been printed, press "G" to draw graph of vector/ phasor. Line 540 - display looks best if quadrant markers 1 - 4 are in inverse video.

Beware of vectors that go off the screen; if you go off to the right or top of the screen, the program stops - enter RETURN to get back to the program. If you go off to the left or the bottom, the vector reflects back - an interesting oddity on the ZX81/TS1000 (PLOT ignores the signs of the argument.) Maximum X value is +/-30, max. Y value is +/-20.

When the display is complete, press any key but BREAK for another problem.

Greyplot Dual Display Technique

Here's a way to use the grey graphics characters to plot data, much like using the PLOT (or UNPLOT) command. Since each character is a full PRINT position wide (=2 PLOT positions) the horizontal resolution is 32 (compared with 64 using PLOT). The vertical resolution though is the same, 44. The usefulness of this is that it lets you plot two different things on the same set of axes.

The effect is especially impressive when grey and white plots are made on a dark background, as in the GREYPLOT1 listing below. The display shows a unit sine wave (in grey) and its square (in white) against a black background.

The heart of the program is the subroutine at lines 40 - 80. Since it is an iterative subroutine (repeatedly used over and over) it is placed near the beginning of the instruction file to reduce the time required to run. To "plot" in grey, call the subroutine with GOSUB 40. To plot (or actually UNPLOT) in white, use GOSUB 50. Generally, when plotting in two shades, you will want to plot the grey one first, in case it gets overwritten; the effect is more pleasing to the eye when crossover points are in the stronger shade (black or white.) Before calling the subroutine, load the vertical value (0 to 43, just like PLOT) into Y, and load the horizontal value (0 to 31) into H (horizontally it behaves like PRINT AT or TAB.)

The program to demonstrate the routine follows. To modify it so that it prints grey and black on a white background, delete lines 110,120 and 130 and change the following lines:

```
40 LET L$="graphic on D"
45 LET H$="graphic on S"
50 LET L$="graphic on 6"
55 LET H$="graphic on 7"
150 LET H$=CHR$(27-4*(H/5=INT
(H/5))
```

```
10 GOTO 100
40 LET L$="█"
45 LET H$="█"
47 GOTO 50
50 LET L$="█"
55 LET H$="█"
60 IF Y/2-INT (Y/2)>.01 THEN L
ET L$=H$
70 PRINT AT 21-INT (Y/2),H;L$
80 RETURN
90 SAVE "GREYPLOT█"
100 CLS
110 FOR H=0 TO 20
120 PRINT "█"
130 NEXT H
140 FOR H=0 TO 30
150 LET H$=CHR$(155-4*(H/5=INT
(H/5)))
160 PRINT AT 10,H;H$
170 IF H<21 THEN PRINT AT H,0;H
$
180 NEXT H
190 PRINT AT 0,1;"+";AT 11,1;"
0";TAB 15;"PI";TAB 29;"2PI";AT 2
0,1;"-1"
200 LET A$="SIN (H*PI/15)"
210 FOR H=0 TO 30
215 LET Z=VAL A$
220 LET Y=22+INT (20*Z+.5)
230 GOSUB 40
240 LET Y=22+INT (20*Z*Z+.5)
250 GOSUB 50
260 NEXT H
```

When you ran GREYPLOT1, you may have noticed that at two points, just before and after the first peak, a grey spot is plotted and then erased by the white (or black) spot, even though the new spot is in the next lower position. This is because when printing the second character, it is in the same PRINT position as the grey character. We can PEEK the appropriate location in the display file to find out if there is already a half-grey character there, and if so make the appropriate correction to L\$ before we print it (or rather POKE - since we already have the location in L, why not?)

GREYPLOT2

Make the following changes to the GREYPLOT1 listing:

```
5 LET DF=PEEK 16396+256*PEEK
16397
62 LET L=DF+33*(21-INT (Y/2))+
H+1
65 IF PEEK L=138 AND L$="gr.7"
THEN LET L$="gr.S"
67 IF PEEK L=137 AND L$="gr.6"
THEN LET L$="gr.D"
70 POKE L,CODE L$
```

Now all is as it should be, and the white spot doesn't wipe out any grey ones in the next place. One warning - when using this method of addressing the display file, be sure to execute the line that looks up the display file start (5 in this case) after making program changes. Otherwise you'll POKE some wrong things, with resulting goofy behavior or crashes. Also, when using a machine with RAMTOP at 3.25K or less (as with a 2K machine) be sure to pad out the display file first; e.g. for a white background,

```
110 FOR H=0 TO 21
120 PRINT AT H,31;" "
130 NEXT A
```

Unless you're certain that the data won't exceed the limits of H and Y, as in the example, you should include traps in case out-of-range numbers are sent into the subroutine. Example: 61 IF H>31 OR H<0 OR Y>43 OR Y<0 THEN RETURN would do the trick. Otherwise, the POKE will change values in the program or variables area instead!

TRIPLLOT

If we take all this to extremes, we might come up with something like the following listing. A sine wave is plotted in grey, and a decaying exponential in small black spots. Their product, a damped sinusoid, is shown with large black spots. The PLOT command in line 270 wipes out adjacent grey spots, much like GREYPLOT1 did. Can you work out a way around this?


```

5 LET DF=PEEK 16396+256*PEEK
16397
10 GOTO 100
40 LET L$=""
45 LET H$=""
47 GOTO 60
50 LET L$=""
55 LET H$=""
60 IF Y/2-INT (Y/2)>.01 THEN L
ET L$=H$
62 LET L=DF+33*(21-INT (Y/2))+
H+1
65 IF PEEK L=10 AND L$="" THE
N LET L$=""
67 IF PEEK L=9 AND L$="" THEN
LET L$=""
70 POKE L,CODE L$
80 RETURN
90 SAVE "TRIPLOC"
95 RUN
100 CLS
110 FOR H=0 TO 21
120 PRINT AT H,01," "
130 NEXT H
140 FOR H=0 TO 30
150 LET H$=CHR$ (27-4*(H/5=INT
(H/5)))
160 PRINT AT 10,H:H$
170 IF H<21 THEN PRINT AT H,0;H
$
180 NEXT H
190 PRINT AT 0,1;"+"(AT 11,1)"
0";TAB 15;"PI";TAB 29;"2PI";AT 2
0,1;"-1"

```

```

200 LET A$="SIN (H*PI/10)"
205 LET B$="EXP (-H/15)"
210 FOR H=0 TO 30
215 LET Z1=VAL A$
216 LET Z2=VAL B$
220 LET Y=22+INT (20*Z1+.5)
230 GOSUB 40
235 LET Y=22+INT (20*Z1*Z2+.5)
250 GOSUB 50
260 LET Y=22+INT (20*Z2+.5)
270 PLOT 2,H,Y
280 NEXT H

```



MATH DEVELOPMENT PROGRAM

Before we get into the "main course" for this issue, I would like to present a brief discussion of what "Simultaneous Linear Equations" are, for the benefit of those who may not be familiar with these concepts, or need to brush up a hazy memory from high-school or college days. It's not intended as a course in S.L.E., as we just don't have the space for such an undertaking. I do hope to give enough information to spur your interest to do as Mr. Bent suggests and bone up on the subject using a good text or even an outline book if all you need is a refresher.

Simultaneous WHAT?

Let's start out with a simple mathematical relation expressed in words. Suppose someone told you he bought 20 apples and 5 mangos for \$5.75. You could not deduce much about the price of apples and mangos as there is an infinite number of possibilities (assuming that fractional cent values are allowed). But suppose he also told you that had he bought 6 apples and 10 mangos it would have cost \$6.40. Ah, now you can sit down with pencil and paper and figure out how much each costs. Before you read on, try to work out the solution to this little problem.

As it turns out, the word problem above can very easily be converted to algebraic equations (how convenient!)

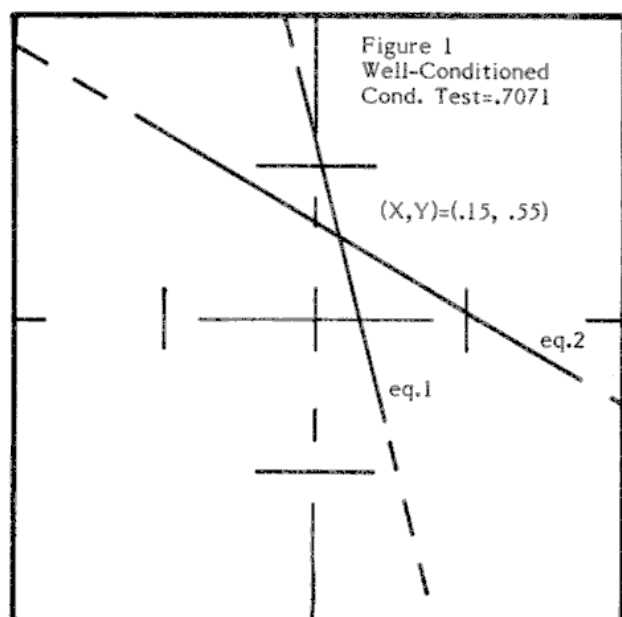
$$20 \cdot X + 5 \cdot Y = 5.75 \text{ and}$$

$$6 \cdot X + 10 \cdot Y = 6.40$$

where X is the price of apples and Y is the price of mangos.

This pair of equations is termed a "system" of equations. In order for both equations to hold true at the same time (hence "simultaneous"), the values for X and Y must "satisfy" both; i.e., when we plug them into either equation we should have both sides of the equal sign the same. Put the values you found into the equations as a check on your work; both equations should balance. (If you've forgotten how to solve this type of system, here's a hint: multiply each term in the first equation by 2, then subtract the second equation term by term from the first. This results in an equation from which Y has been eliminated, and you should have no trouble finding X=.15 and then put that value back into either of the original equations and get Y=.55.)

OK so far, but why "linear?" Linear means having to do with straight lines, and linear equations are ones that produce straight lines when graphed. As it turns out, this is only true of equations in which the variables are not raised to powers or have other functional operations (e.g. SIN, ATN, LN, etc.) performed on them. In this installment we will only deal with this type of system. If we graph our fruity problem above, we come up with a plot like Fig. 1. Line 1 represents all the combinations of x and y which satisfy the first equation, and line two is all the points which



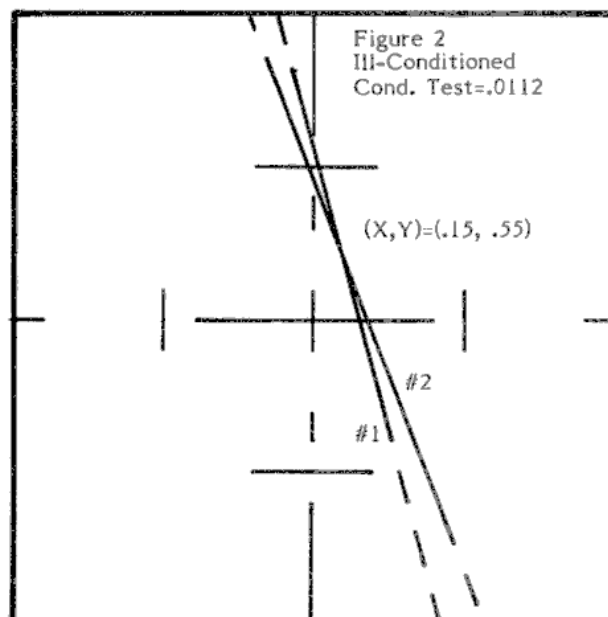
satisfy the second. The point where they intersect is the solution - the only point which simultaneously satisfies both equations.

At this point, let's assume that the second equation were $21X + 5Y = 5.90$ instead. The system still gives the same solution, as you can verify. A graph of this modified system is shown in Fig. 2. Again there's one discrete solution, but in this case the lines are very nearly parallel. Such a system is called "ill-conditioned," and is less likely to give accurate results when using a computer. Just as it's harder to discern the intersection point on the graph, so also the computer has difficulty resolving the exact solution. The program includes a routine that checks for ill-conditioning and yields a number between 0 (parallel lines or the same line and possessing no discrete solution) and 1 (perpendicular lines, and most accurate) to rate the system.

So big deal. We don't need a computer to solve simple systems like this. The rub comes when we try to solve systems with more than two variables - try solving a six-variable system sometime! Note that for a solution we must have at least as many equations as we have unknowns, so you can see that things rapidly get out of hand. What we need is a systematic approach the computer can just crunch away at. Such systematic approaches were developed long before computers were, so don't get the idea that all this is a new development. With a computer, though, even the huge task of solving 10th order S.E. becomes routine.

The first step is to convert our system of equations into an array called a matrix. Matrices are mathematical entities that can be operated on according to certain rules. Study Mr. Bent's multiplication example to see why we can express our system as follows:

$$\begin{bmatrix} 20 & 5 \\ 6 & 10 \end{bmatrix} * \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} 5.75 \\ 6.40 \end{bmatrix}$$



We thus have two arrays of numbers, one on either side of the equal sign, that specify the system. The matrix on the left side contains the multipliers (coefficients) of the unknowns, and the one on the right contains the constants each equation adds up to. For convenience, we can combine the two matrices into one 2×3 matrix, as this makes it easier to input the numbers. Just remember that the last column is the constants, the others are the coefficients.

Now that we have our matrix, we need some way to boil it down into answers. One way is by breaking it down into "determinants," which is another form of array. Unlike a matrix, though, a determinant boils down to a single number, when evaluated according to certain simple rules. The bigger it is, the more calculations it takes, but you'll always end up with a number that can be used to obtain a solution. To keep this distinction in view, we'll indicate matrices with brackets, and determinants with straight lines. Arrays that are essentially a "butt-splice" of two matrices -- i.e. the way you input the numbers to the computer, are shown with curved brackets as they are neither matrices nor determinants but a shortcut for convenience. These will usually be $N * N+1$ in dimension.

I hope this has whetted your appetite for the article. A few short notes on the program: Enter RUN to start, push F or S for fast or slow mode. The main menu appears, options 1-3 allow matrix operations, 4 evaluates a determinant, and 5 goes on to the main S.L.E. portion. Selecting this gives an SE menu that is a veritable smorgasbord of possibilities. I particularly like combination plate #7, I think you will too. After completing analysis by Gauss Elimination and/or Gauss Seidel any size, you can repeat the problem using Cramers Rule by selecting 5, then 2, and pressing "N" on "NEW DATA" prompt. This is the only program I've seen allowing all three methods, even though CR any size "cheats" by using GE to do the dirty work. For this reason it usually gives the same results as GE (actually

results are more accurate, but the time scale is considerably larger), but I added it to demonstrate row-swapping of the last row with the others to get the determinants for each answer.

And now, heere's Thomas!

Part 1

By Thomas Bent

First, a hearty hello to all you ZX/TSers out there! I would like to say a few words about myself before going too far. My computer background consists of a couple years of Basic and Fortran with Pascal as a new addition. For those of you familiar with DEC equipment, some Focal language also. My job doesn't consist of a lot of programming, but the need to use a computer to solve problems for myself (and my boss) sent me in search of this programming tool. Since I do quite a bit of materials testing, I do a fair amount of set-up and program design on an assortment of computers. I am also a professional pilot holding a commercial, multi-engine rating. Should you wish to contact me for more info or even to tell me my blunders, my home address is:

Thomas Bent
9016 Flicker Place
Columbia, MD 21045

Phone: (301) 730-7187

Many of those who initially purchase the ZX/TS computer buy it with little knowledge of how computers work. Most of the rest, who know how they work, can't afford anything more expensive. Then there are those who hold out for a more expensive computer simply because they can't believe that a \$30.00 computer can do what they want. Heh-heh...

I have been pleasantly surprised with my Sinclair in the past two and a half years, and I look forward to a computerized world in which I know I will be able to hold up my end of the load. What I hope to do in a series of upcoming articles is to bring newcomers "up to speed" in some basic techniques of programming math routines on the ZX/TS, so that you won't have to shy away from a project simply for a lack of knowledge on how to program these routines. Those who are familiar with this material can incorporate the routines into their own programs conveniently.

What I plan to cover will include but not be limited to more integration, numerical differentiation (polynomial and real-time,) curve-fitting of various types, finding roots of equations, and simultaneous equations. Some of these techniques are hard to understand at first, but very easy to program. Please try to follow the development of the first subject, Simultaneous Linear Equations, because these routines will be used as subroutines in many programs that lie ahead.

Solving Simultaneous Linear Equations

Most high-school and college students know how to solve linear simultaneous equations (sim. eq.) and I'm sure they will agree that they have better things to do with their time. This first part will provide some useful programs for solving these types of problems, and will greatly speed up that engineering and/or math homework.

Before we start, let's get one thing straight; all computers are notorious liars. Apple, Atari and VIC computers (6502 chip) do not handle floating point mathematics well. One reason that the ZX/TS is much slower when grinding numbers is because it has much better resolution (5-byte floating point as opposed to 4-byte for the others, or two extra decimal places accuracy). However if you want speed then you can always get a compiler (e.g. Bob Berch, 19 Jacques St., Rochester, NY 14620) and speed up those "integer" calculations anyway. REMEMBER!! There are many large holes in the computer number line and many numbers are stored inaccurately (e.g. Pi, .2, 1.3). This is quantizing error. Addition, subtraction, etc. are also done inaccurately (try taking the cube root [or square root] of eight, cube it [square it], and subtract 8 on your calculator). This is roundoff error. Heaven forbid you should add two inaccurate numbers many times and say, "the answer is...."!! One last point, if your data is accurate to two decimal places then your answer after several calculations certainly is no better. Throw away all those insignificant figures, they do you more harm than good.

Alright now, on with the show.

The good news is, this series will provide a complete computer math program and hopefully some understanding behind how the routines work. The bad news is that the material tends to be rather dry (and may require some WORK). The laughs will come later when you see how easy the homework becomes. The future chapters of this report will be integrated with this first section. I think if you look at the listing, you'll see that expansion should be very easy.

The Sim. Eq. techniques (S.E.) we will delve into this time around are Gauss-Seidel, Cramer's Rule, and Gauss elimination. This is a tough subject to explain and to grasp, fortunately it is easier to program. It may require some extra homework, but if there are any questions don't hesitate to write (this still holds-ed.) If S.E. are understood then many upcoming topics will be a snap.

Manipulating matrices is the main scientific use of computers. You can add and subtract matrices of the same size, or multiply them by a constant. You can also multiply square or rectangular matrices where the number of columns of the first matrix equals the number of rows of the second. Multiplying matrices is only tricky at first, but as you can see in the program, this is done with three loops and one calculation line. Please take the time to work through the given multiplication example so that the following S.E. routines will be more familiar and easier to debug.

Addition

$$\begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix} + \begin{bmatrix} 9 & 8 \\ 6 & 5 \end{bmatrix} = \begin{bmatrix} 10 & 10 \\ 10 & 10 \end{bmatrix}$$

Subtraction

$$\begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix} - \begin{bmatrix} 9 & 8 \\ 6 & 5 \end{bmatrix} = \begin{bmatrix} -8 & -6 \\ -2 & 0 \end{bmatrix}$$

Multiplication by
a constant

$$2 * \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$$

You multiply rectangular matrices as follows:

$$A \ 4 \times 2 * 2 \times 4 = 4 \times 4$$

$$A \ 2 \times 4 * 4 \times 2 = 2 \times 2$$

I have worked out an example of the first case to show all the calculations involved. In the program, you input the size of matrix A and the computer only accepts input for the number of columns of matrix B. The product, matrix C has as many rows as A and as many columns as B.

$$\begin{bmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 7 \\ 4 & 8 \end{bmatrix} * \begin{bmatrix} 2 & 4 & 6 & 8 \\ 3 & 5 & 7 & 9 \end{bmatrix} =$$

$$\begin{bmatrix} 1*2+5*3 & 1*4+5*5 & 1*6+5*7 & 1*8+5*9 \\ 2*2+6*3 & 2*4+6*5 & 2*6+6*7 & 2*8+6*9 \\ 3*2+7*3 & 3*4+7*5 & 3*6+7*7 & 3*8+7*9 \\ 4*2+8*3 & 4*4+8*5 & 4*6+8*7 & 4*8+8*9 \end{bmatrix}$$

$$= \begin{bmatrix} 17 & 29 & 41 & 53 \\ 22 & 38 & 54 & 72 \\ 27 & 47 & 67 & 87 \\ 32 & 56 & 80 & 104 \end{bmatrix}$$

It is much easier for the ZX/TS to do this kind of grunt work, and I think you'll agree.

Determinants

Determinants (det) are SQUARE arrays of any size, e.g. 2x2, 10x10, with some unique properties. A det is a single number in the form of an array of numbers. An Nth order det generates a predetermined number of calculations to solve, namely N factorial (N!) times (N-1). A 3rd order (3x3) det generates 12 calculations (3*2*1)*(3-1). A 10th order det generates 10!*9 or in excess of 32,600,000 calculations! There are identities (det = 1) and inverse dets, among others. Please refer to a good math book or even flip through a Schaum's outline book at your nearest book store for the properties of dets. Two important properties we must program for are:

1. If you interchange two rows of a det, the absolute value does not change, but the sign is reversed.

2. Adding one row of a det to another does not change the value of the determinant.

Evaluating a small det is easy and is done as follows:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = A*D - B*C \quad \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix} = 2*8 - 4*6 = -8$$

In array form it looks like this:

$$\begin{bmatrix} A(1,1) & A(1,2) \\ A(2,1) & A(2,2) \end{bmatrix} = A(1,1)*A(2,2) - A(2,1)*A(1,2) \\ \text{A SINGLE NUMBER}$$

Larger ones just get more cumbersome and tougher to keep straight.

$$\begin{bmatrix} 2 & 4 & 6 \\ 3 & 5 & 7 \\ 4 & 6 & 8 \end{bmatrix} = 2 * \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix} - 4 * \begin{bmatrix} 3 & 7 \\ 4 & 8 \end{bmatrix} + 6 * \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix} = 0$$

The large det just gets broken down into smaller units and manipulated. By the way, if one row is a multiple of another row, the determinant = 0.

Gauss Elimination

(Ye Olde Standard)

Taking what has just been discussed, we will transform our det (since we know some key properties) to reduce the astronomical number of calculations to something more manageable. In order to reduce round-off error, we should insure that the diagonal terms [A(1,1), A(2,2), etc.] contain the largest absolute value in their respective columns, for the terms below them. By this I mean that after evaluating col 1 row 1, leave that row alone [the col values below A(1,1) will all be zero.] Of course the last row gets whatever values are left.

What happens in the program is, the det is INPUT and col 1 is checked for the largest term. A zero term is checked for as this would mean an infinite number of solutions. The row with the largest value in col 1 is swapped with row 1. The new row 1 is multiplied by a different constant for each remaining row, so that every value in col 1 below the first term is zero when row 1 is subtracted from the row in question. Row 1 is now complete, as is col 1. Our "Pivot" now moves diagonally A(2,2), and the same trick is done again on the smaller array and so on until the last row (where there is only one term left). You must keep track of the number of times you swap rows (sign change) and simply multiply all of the diagonal terms together with the value of the sign (+ or -1) and voila! A 10x10 array takes less than 1000 calculations. A 6x6 array would not be uncommon for an engineer or engineering student. Most of us usually only see a 3x3 (because of the complexity and grunt work).

Simultaneous Equations (SE) will typically be 3x4 [or generally, Nx(N+1)] arrays and are handled much the same as determinants. Now we no longer have one number to deal with and must "back-calculate" the values that will solve the system of equations in question. The matrix is reduced the same way as the det is, except now we rearrange (put everything on the other side of the equality) our array to produce the values for X and Y [X(1) and X(2).]

The general form is:

$$\begin{aligned} AX + BY &= E \\ CX + DY &= F \end{aligned}$$

A, B, C, D, E and F are constants that are entered. X and Y we must solve for.

After reducing the matrix by multiplying row 1 by C/A and subtracting from the second row, the CX term cancels out and the second row becomes DY - CB/AY = F (the first row doesn't change value.) The computer can quickly solve for Y and put that value in equation 1 and back out the value of X. There are no sign changes in SE to keep track of.

The computer will look for equations like this to solve for:

$$\begin{aligned} A(1,1) + A(1,2) + A(1,3) &= A(1,4) \\ A(2,1) + A(2,2) + A(2,3) &= A(2,4) \\ A(3,1) + A(3,2) + A(3,3) &= A(3,4) \end{aligned}$$

In the programs we dimension A(M,N) where M is the number of rows and N is the number of columns. This will give generality for any size array you have to solve for. X is dimensioned by M also, as linear SE are only valid for one unknown per equation. The computer will respond with one X for each column, that is, the answer for column 1 is X(1), etc. and that X is supposed to satisfy all M equations. (Hope and pray!)

Cramer's Rule

(The Proverbial Hand Job)

Cramer's Rule is very easy to program for small arrays (2x3), but gets tougher quickly thereafter. For 2 equations & 2 unknowns, in fact, it consists of only 3 program lines. Cramer's Rule (CR) breaks the matrix into determinants (only one number) and solves for each unknown independently. This method is very close to how you solve linear SE by hand. Look at the following equations:

$$\begin{aligned} AX + BY &= C \\ DX + EY &= F \end{aligned}$$

Multiply the top equation by E, the bottom equation by B and subtract the second equation from the first.

$$\begin{aligned} +(EAX + EBY &= EC) \\ -(BDX + BEY &= BF) \\ \hline \end{aligned}$$

$$EAX - BDX = EC - BF$$

Then consolidate X and solve.

$$\begin{aligned} X * (EA - BD) &= EC - BF \text{ or} \\ X &= (EC - BF) / (EA - BD) \end{aligned}$$

If this does not look familiar to you at this point, have no fear. We shall repeat the same thing again below the way the computer sees it.

$$\begin{aligned} AX + BY &= C \\ DX + EY &= F \\ X &= (C * E - B * F) / (A * E - B * D) \text{ and} \\ Y &= (A * F - C * D) / (A * E - B * D) \end{aligned}$$

or,

$$X = (\text{DET } X) / (\text{DET}) \quad Y = (\text{DET } Y) / (\text{DET})$$

where

$$\text{DET} = \begin{vmatrix} A & B \\ D & E \end{vmatrix} = AE - BD$$

$$\text{DETX} = \begin{vmatrix} C & B \\ F & E \end{vmatrix} = CE - FB$$

$$\text{DETY} = \begin{vmatrix} A & C \\ D & F \end{vmatrix} = AF - CB$$

If you look back at the section on determinants, you should see the connection. The denominator is the same for both X and Y, and the numerator swaps terms. Watch that the denominator is not zero as the solution is undefined. How this works is easy to see in a 3x4, but is tougher to program directly, as in the case of the 2x3.

$$\begin{aligned} A1 + B1 + C1 &= Y1 \\ A2 + B2 + C2 &= Y2 \\ A3 + B3 + C3 &= Y3 \end{aligned}$$

The three answers are as follows:

$$X1 = \frac{\begin{bmatrix} Y1 & B1 & C1 \\ Y2 & B2 & C2 \\ Y3 & B3 & C3 \end{bmatrix}}{\begin{bmatrix} A1 & B1 & C1 \\ A2 & B2 & C2 \\ A3 & B3 & C3 \end{bmatrix}} \quad X2 = \frac{\begin{bmatrix} A1 & Y1 & C1 \\ A2 & Y2 & C2 \\ A3 & Y3 & C3 \end{bmatrix}}{\begin{bmatrix} A1 & B1 & C1 \\ A2 & B2 & C2 \\ A3 & B3 & C3 \end{bmatrix}} \quad X3 = \frac{\begin{bmatrix} A1 & B1 & Y1 \\ A2 & B2 & Y2 \\ A3 & B3 & Y3 \end{bmatrix}}{\begin{bmatrix} A1 & B1 & C1 \\ A2 & B2 & C2 \\ A3 & B3 & C3 \end{bmatrix}}$$

I prefer Gauss Elimination when solving for any system of higher order than 2x3, as CR requires a lot more calculations (takes longer and accuracy suffers) and also requires more memory space.

Gauss Seidel

(Miracle cure or just another rip-off?)

This method is designed for a computer although it is an old technique. What it does is start with a guess at what the value is, plugs and chugs and goes back to the beginning with a new guess (the value it just solved for). Sounds great! Only one drawback; it doesn't always work. However, we can stack the odds in our favor. Gauss-Seidel (GS) is guaranteed to work if the value of the diagonal terms are greater than the sum of the terms in the row plus the sum of the terms in the column. Another way to help make this work will be to begin with a different starting value if necessary. GS is very memory efficient and is not susceptible to round-off error like the other techniques. Each iteration is just like starting over, if the present answer is not good enough then go back and try again. You can also watch the computer approach the answer rather than sitting and waiting.

Should GS diverge or jump around, simply BREAK in and assign X values directly (LET X(1) = -10 or some other value), then CONT to see if that helps.

If that doesn't work, then BREAK and RETURN to try another way. Initially all X values will be assigned zero. The equations are swapped using the GE subroutine (but not reduced), unless you input the equations in the proper order, in which case the subroutine can be bypassed (saving memory). The computer will manipulate the equations to simulate the following:

$$AX(1) + BX(2) = C$$

$$DX(1) + EX(2) = F$$

Then,

$$X(1) = (C - BX(2)) / A$$

$$X(2) = (F - DX(1)) / E$$

and so on.

Actually, the X term being evaluated is on both sides of the equation, but is assigned to zero every time through the loop. There must be an outside loop controlling the maximum number of iterations in case of divergence. Normally, 30 iterations will be more than enough for 6-place accuracy, but 100 may be necessary depending on how ill-conditioned the system of equations is (see the graphs).

To test for ill-conditioning, you "normalize" the determinant of the system of equations. This means that you square each term in the row, add them up and divide each term in the row by the sum of the squares (more on this next time). GOSUB GE to evaluate the determinant and you have a number less than one. If it is less than .2 then you may have trouble trying to solve your SE by any other method than by hand (but not necessarily). If you are concerned about the accuracy of your answers then try changing your matrix slightly and see if your answers vary largely. If so then get out your pencil and paper.

Have fun!!!

'SE' Annotated Listing

This material is based on the kernel developed in vol. 1:2 and 1:3, and 32K up is recommended. 16K owners will have to break this material into sections for each task and SAVE separately; start with the SE kernel and add only those PLOT/FIT and INTEGRATION routines you'll need.

```
10 REM SE VERSION 3.2 4/1/84
34 GOTO 1000
38 REM INPUT XXXXXXXXXX
42 PRINT "XXXXXXXXXXXX"
46 INPUT M
50 LET DP=M
54 DIM P(M)
58 DIM U(M)
62 DIM V(M)
66 DIM Y(M)
70 PRINT " INPUT X THEN Y VAL
..
74 FAST
78 FOR I=A TO M
82 PRINT TAB TR;I;" "
86 INPUT P(I)
90 INPUT Y(I)
94 PRINT P(I),Y(I)
98 IF I>18 THEN GOSUB 910
102 NEXT I
106 GOTO 666
110 PRINT " INPUT START / STO
P FOR X THEN Y"
114 DIM E(D)
118 DIM E$(D,D)
122 FOR I=A TO D
126 IF I=A OR I=TR THEN PRINT (
"X " AND I=A)+("Y " AND I=TR)
129 INPUT E$(I)
134 PRINT E$(I),
138 NEXT I
142 RETURN
```

```
;If a word looks like a KEYword, then it IS a KEYword
;1000 is main menu

;Data point entry routine for plotting and fitting
;How many points
;M and DP = # of points, M also= # of rows in
simultaneous equation solution
;P = X data array
;U = X fit array
;V = Y fit array
;Y = Y data array
;Having separate arrays for data and fit will insure
that your data won't get crunched somewhere
;Input X and Y as pairs. To correct bad data, GOTO 4000
and print your data. Break at the right point and in
the direct mode, LET P (or Y)=nn, ENTER and RETURN.
;GOTO main plot and fit menu
;Give graph limits. This is necessary ONLY on the first
time unless you want to change your limits. It is
necessary to initialize all of the graphing FLAGS for
future use
;D=4. E is the manipulated variable and E$ is printed
as graph limits. I suggest values like 1E1 for 10
2E2 FOR 200 1E-1 FOR 0.1 ETC., up to 4 characters.
```

```

146 FOR I=A TO D
150 LET E(I)=VAL E$(I)
154 NEXT I
158 RETURN
162 PRINT " INPUT TITLE ";
166 INPUT T$
170 PRINT T$
174 PRINT " INPUT X LABEL ";
178 INPUT X$
182 PRINT X$
186 PRINT " INPUT Y LABEL ";
190 INPUT Y$
194 PRINT Y$
198 RETURN
202 PRINT " INPUT X LOG LIN ";
206 INPUT XL
210 PRINT " INPUT Y LOG LIN ";
214 INPUT YL
218 RETURN
222 LET E(A)=LN E(A)/K
226 LET E(TW)=LN E(TW)/K
230 RETURN
234 LET E(TR)=LN E(TR)/K
238 LET E(D)=LN E(D)/K
242 RETURN
246 REM SET SCALE
250 DIM O(M)
254 DIM Q(M)
258 CLS
262 PRINT " LABEL GRAPH 1=0 YES
266 INPUT SL

270 IF SL THEN GOSUB 162
274 CLS
278 PRINT " SET SCALE 1=0 YES
282 INPUT SS
286 IF SS THEN GOSUB 202
290 PRINT " SET HIGH LOW LIMITS
294 INPUT SSL
298 IF SSL THEN GOSUB 110
302 PRINT " POINTS CONNECTED
306 INPUT CN
310 PRINT " PLOT CURVE FIT 1=0 YES
314 INPUT CF
318 PRINT " LIMITS OK 1=0 YES
322 INPUT Z
326 CLS
330 IF NOT Z THEN GOTO 246
338 GOSUB 146
342 LET K=LN 10
346 IF NOT XL THEN GOTO 366
350 GOSUB 222
354 FOR I=A TO M
358 LET U(I)=LN P(I)/K
362 NEXT I
366 IF XL THEN GOTO 382
370 FOR I=A TO M
374 LET U(I)=P(I)
378 NEXT I
382 IF NOT YL THEN GOTO 402
386 GOSUB 234
390 FOR I=A TO M
394 LET V(I)=LN Y(I)/K
398 NEXT I
402 IF YL THEN GOTO 418
406 FOR I=A TO M
410 LET V(I)=Y(I)
414 NEXT I
418 REM SET NO
422 LET DX=(E(TW)-E(A))/56
426 LET DY=(E(D)-E(TR))/40
430 LET PL=A
434 FOR I=A TO M
438 LET TX=(U(I)-E(A))/DX
442 IF TX<Z0 OR TX>57 THEN GOTO
446 LET TY=(V(I)-E(TR))/DY
450 IF TY<Z0 OR TY>40 THEN GOTO
454 LET O(PL)=INT (TX+7.5)
458 LET Q(PL)=INT (TY+3.5)
462 LET PL=PL+A
466 NEXT I
470 FOR I=TR TO 63
474 IF I<44 THEN PLOT 6,I
478 IF I>5 THEN PLOT I,TW
482 NEXT I
486 IF NOT SL THEN GOTO 518
490 LET T=10-LEN Y$/TW
494 FOR I=A TO LEN Y$
498 PRINT AT T,A;Y$(I)
502 LET T=T+A
506 NEXT I

```

;Get new variable each time, in case limits are changed

;Labeling does not have to be done until you are ready for a hard copy of your graph

;Title
;X Label
;Y Label

;Set LOG/LIN scale first time through unless you want to change it.

;X LOG scale-- XL=1
;X Linear----- XL=0
;Y LOG--YL=1
;Y LIN--YL=0

;Convert E to LOG scale if necessary, K=LN10 (constant) for base 10 calculations
;A=1 TW=2--X axis

;TR=3 D=4--Y axis

;O=X PLOT points
;Q=Y PLOT points

;This is the beginning of the prompt section. It will call the previous subroutines to set the parameters specified. Because of the use of Flags, 1= yes and 0= no. All flags are two characters in length. This will help you see which ones are flags.

;Label graph 1=Y 0=N, not necessary

;SL=subroutine Label

;Set Scale, answer 1 on the first time through
;SS=subroutine Scale

;Set high and low limits, answer 1 first time
;SSL=subroutine Set Limits

;Points connected, 1 or 0 at any time

;CN=connect

;Plot curve fit, only after you have FIT and scaled your data for your graph limits. This program as listed does not have an AUTO PLOT mode yet. (There is room at 750 to 900 for all of the LET's and GOSUB's. See line 1260 to 1400 for more insight)

;CF=Curve fit

;Limits OK? If not then go back to line 246

;K=LN10--log constant

;X LOG. This area sets the correct scale.

;X LIN

;Y LOG

;Y LIN

;This area defines the PLOT limits, STEP and gets all of the points to plot that will fit in the limits that you gave back at line 110. If you have hi res or a 2068 then change 56 (X) to something big (220?) and 40 (Y) to 180?

;DX=X step DY= Y step

;PL=loop counter and will = one more than the number of points that will actually be plotted

;TX and TY= temporary X and Y. They will be put in O and Q if and only if they are on scale else they are forgotten.

;Z0=zero, hi res change 57 and 40 to something big

;Fill plot arrays (integers) point by point

;Line 470 is the re-entry point to quickly replot your data
;PLOT axes. Hires change 44, 5, 6, 63, and TR (=3)

;No label then skip

;This area will center your graph labels down and across


```

1630 REM GAUSS ELIMINATION
1640 DIM A(M,N)
1650 FAST
1660 FOR I=A TO M
1670 FOR J=A TO N
1680 LET A(I,J)=Z(I,J)
1690 NEXT J
1700 NEXT I
1710 SLOW
1720 RETURN
1730 REM GAUSS SEIDEL
1740 GOSUB 1400
1750 DIM X(M)
1760 REM GAUSS SEIDEL
1770 FAST
1780 FOR I=A TO M-A
1790 LET B=Z0
1800 FOR J=I TO M
1810 LET P=ABS A(J,I)
1820 IF P>B THEN LET L=J
1830 IF P>B THEN LET B=P
1840 NEXT J
1850 IF NOT B THEN STOP
1860 IF IL OR I=L THEN GOTO 1930
1870 LET S=-S
1880 FOR J=A TO N
1890 LET T=A(I,J)
1900 LET A(I,J)=A(L,J)
1910 LET A(L,J)=T
1920 NEXT J
1930 IF GS THEN NEXT I
1940 IF GS THEN RETURN
1950 REM GAUSS SEIDEL
1960 LET NLINE=I+A
1970 FOR J=NLINE TO M
1980 LET C=A(J,I)/A(I,I)
1990 FOR K=I TO N
2000 LET A(J,K)=A(J,K)-C*A(I,K)
2010 NEXT K
2020 NEXT J
2030 NEXT I
2040 IF DE OR IL THEN RETURN
2050 REM GAUSS SEIDEL
2060 FOR I=M TO A STEP -A
2070 LET BK=A(I,N)
2080 IF I=M THEN GOTO 2120
2090 FOR J=M TO I STEP -A
2100 LET BK=BK-A(I,J)*X(J)
2110 NEXT J
2120 LET X(I)=BK/A(I,I)
2130 NEXT I
2135 IF IP THEN RETURN
2140 PRINT "RESULTS BY GAUSS ELIMINATION:"
2150 SLOW
2160 FOR I=A TO M
2170 PRINT TAB D;"X ";I,X(I)
2180 NEXT I
2190 RETURN
2200 REM GAUSS SEIDEL
2210 PRINT "INPUT ACCURACY DESIRED"
2220 INPUT ER
2230 CLS
2240 GOSUB 1400
2250 LET GS=A
2260 PRINT AT 19,D;"GAUSS SEIDEL ITERATIONS"
2270 GOSUB 1760
2280 LET GS=NOT GS
2290 DIM X(M)
2300 DIM Y(M)
2310 FAST
2320 FOR K=A TO 100
2330 GOSUB 910
2340 FOR I=A TO M
2350 LET T=Z0
2360 LET X(I)=T
2370 FOR J=A TO M
2380 LET T=T+A(I,J)*X(J)
2390 NEXT J
2400 LET X(I)=(A(I,N)-T)/A(I,I)
2410 NEXT I
2420 LET T=Z0
2430 PRINT K;
2440 FOR I=A TO M
2450 PRINT TAB D;"X ";I,X(I)
2460 GOSUB 910
2470 LET Y(I)=X(I)-Y(I)
2480 IF ABS Y(I)<=ER THEN LET T=T+A
2490 LET Y(I)=X(I)
2500 NEXT I
2510 PAUSE 100
2520 IF T<>M THEN GOTO 2580
2530 CLS
2540 PRINT "GAUSS SEIDEL CONVERGED", "IN ";K;" ITERATIONS, RESULTS ARE:"
2550 GOSUB 2160
2560 SLOW
2570 RETURN

```

;This is where you get it back.

;Gauss Elim. starts here
;input array
;X holds the solutions
;Swap rows is called by just about every routine in this program. For best accuracy it puts the biggest numbers 1st.

;If B = 0 then you have a problem with your data. No answer.
;IL=ill conditioning check flag. already swapped.
;S=determinant flag not considered otherwise
;Actual 2 DIMension row swap here

;GS = Gauss Seidel (and a few others). Don't reduce array.
;GS has its own method. Just swap rows.

;This reduction destroys the A array. Turns it into a triangle. From whence you can back calculate each X value

;If a determinant or ILL cond. ck. then don't back calculate.
;Figure out the solution (read text)

;if Interpolation then go back there
;Print results from any routine that requires it

;Gauss Seidel is an iterative routine that needs a stopping criterion (ER). The smaller the ER then the smaller the error, but the longer the calculation time. Don't pick an ER less than 1E-8

;Set GS flag & call Gauss Elim. Swap rows

;K= outside loop to prevent overworking your poor little ZX
;Read Text
;Temp. variable
;Zero next calculation

;Sum all X terms*coefficients

;Subtract from Y

;Check to see if all new calculations-old calculations<=ER

;If so then go on and print results

```

2580 NEXT K
2590 PRINT "GAUSS SEIDEL DID NOT
CONVERGE"
2600 SLOW
2610 RETURN
2620 REM CRAMER'S RULE
2630 LET M=TW
2640 LET N=TR
2650 GOSUB 1400
2660 LET CR=A(A,A)*A(TW,TW)-A(TW
,A)*A(A,TW)
2670 IF NOT CR THEN GOTO 3990
2675 DIM X(M)
2680 LET X(A)=(A(A,TR)*A(TW,TW)-
A(TW,TR)*A(A,TW))/CR
2690 LET X(TW)=(A(A,A)*A(TW,TR)-
A(TW,A)*A(A,TR))/CR
2700 PRINT "RESULTS BY CRAMER'S
RULE ARE"
2710 PRINT "X=";X(A);"Y=";X(
TW)
2720 RETURN
2730 REM CRAMER'S RULE
2740 GOSUB 1400
2750 PRINT "INPUT MATRIX ""B""
2760 DIM B(M,N)
2770 FOR I=A TO M
2780 PRINT "INPUT ROW ";I
2790 FOR J=A TO N
2800 INPUT B(I,J)
2810 NEXT J
2820 NEXT I
2830 CLS
2840 DIM C(M,N)
2850 PRINT "ROW COL";TAB 13;"MAT
RICES"
2860 PRINT "I";TAB 9;"A";TAB
16;"B";TAB 25;"C"
2870 FOR I=A TO M
2880 FOR J=A TO N
2890 IF Z=A THEN LET C(I,J)=A(I,
J)+B(I,J)
2900 IF Z=TW THEN LET C(I,J)=A(I,
J)-B(I,J)
2910 PRINT I;TAB D;J;TAB TR*TR;A
(I,J);TAB D*D;B(I,J);TAB 25;C(I,
J)
2920 NEXT J
2930 NEXT I
2940 RETURN
2950 REM CRAMER'S RULE
2960 GOSUB 1400
2970 PRINT "HOW MANY COLS FOR B?"
2980 INPUT P
2990 DIM B(N,P)
3000 FOR I=A TO N
3010 PRINT "INPUT ROW ";I
3020 FOR J=A TO P
3030 INPUT B(I,J)
3040 NEXT J
3050 NEXT I
3060 CLS
3070 DIM C(M,P)
3080 FOR I=A TO M
3090 FOR J=A TO P
3100 FOR K=A TO N
3110 LET C(I,J)=C(I,J)+A(I,K)*B(
K,J)
3120 NEXT K
3130 NEXT J
3140 FOR J=A TO P
3150 PRINT C(I,J);"
3160 NEXT J
3170 PRINT
3180 NEXT I
3190 RETURN
3200 REM DETERMINANT
3210 GOSUB 1400
3220 LET DE=A
3230 GOSUB 1760
3240 LET DE=NOT DE
3250 FOR I=A TO M
3260 LET S=S+A(I,I)
3270 NEXT I
3280 IF CR THEN RETURN
3290 PRINT "THE DETERMINANT = ";
S
3300 RETURN
3310 GOSUB 1400
3320 LET GS=NOT GS
3330 GOSUB 1760
3340 LET GS=NOT GS
3350 REM CRAMER'S RULE
3360 DIM C(M,M)
3370 FOR I=A TO M
3380 LET T=Z0
3390 FOR J=A TO M
3400 LET C(I,J)=A(I,J)
3410 LET T=T+C(I,J)*C(I,J)
3420 NEXT J

```

;Cramer's Rule 2*3 STANDARD

;CR=denominator. It cannot = 0

;The number of rows are determined by array A (see text)

;Sum row A terms*column B terms

;The determinant is determined by the value of the diagonal terms of A (after Gauss E), multiplied together with S

;For Cramer's Rule any size

;This routine determines if there might be a problem with your data and pretty much just computes the angle at which your lines intersect. 1= 90 degrees, .2 or less=borderline solution. 0= parallel lines.

;C stores matrix A and normalizes it <1

```

3430 LET T=SQR T
3440 FOR J=A TO M
3450 LET C(I,J)=C(I,J)/T
3460 NEXT J
3470 NEXT I
3480 IF Z=D+TW THEN GOTO 3510
3490 RETURN
3500 REM GE
3510 FOR I=A TO M
3520 FOR J=A TO M
3530 LET A(I,J)=C(I,J)
3540 NEXT J
3550 NEXT I
3560 LET N=M
3570 LET IL=NOT IL
3580 GOSUB 1760
3590 LET N=M+A
3600 LET IL=NOT IL
3610 FOR I=A TO M
3620 LET S=ABS (S*A(I,I))
3630 NEXT I
3640 PRINT "ILL CONDITIONING T
EST ";S
3650 IF S<A/D THEN PRINT " THE S
YSTEM IS ILL CONDITIONED"
3660 IF S>A/D THEN PRINT "THE S
YSTEM IS OK"
3670 RETURN
3680 REM GE
3690 PRINT " INPUT NEW DATA? Y=1
N=0"
3700 INPUT X
3710 IF X THEN GOSUB 1400
3720 LET CR=A
3730 CLS
3740 GOSUB 1630
3750 LET N=M
3760 FAST
3770 GOSUB 3220
3780 LET DEN=S
3790 PRINT "DENOMINATOR=" ;S
3800 IF NOT S THEN GOTO 3990
3810 DIM X(M)
3820 FOR V=A TO M
3830 LET S=A
3840 LET N=M+A
3850 GOSUB 1630
3860 FOR U=A TO M
3870 LET A(U,V)=A(U,N)
3880 NEXT U
3890 LET N=M
3900 GOSUB 3220
3910 LET X(V)=S/DEN
3920 NEXT V
3930 LET N=M+A
3940 LET CR=NOT PI
3950 SLOW
3960 PRINT "RESULTS BY CR:"
3970 GOSUB 2160
3980 RETURN
3990 PRINT "NO SOLUTION BY CR"
3995 RETURN

```

;Z=6 then get ill array back and compute determinant else CONT

;Fred's Cramer's Rule any size. This uses GE for the tough stuff, but may give slightly better results overall. It takes a lot longer to run

;It computes a determinant for each answer individually

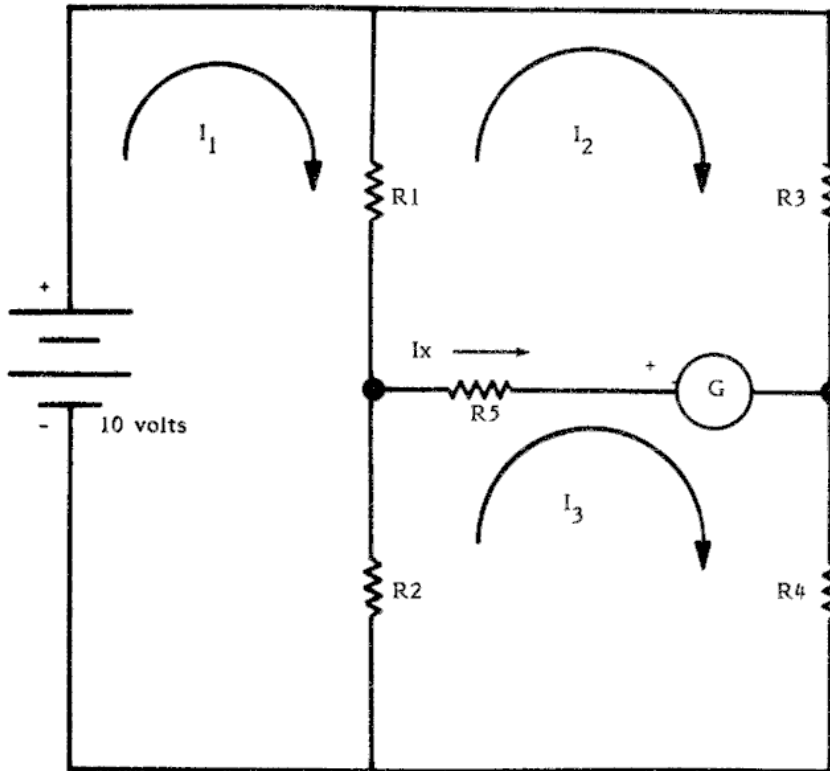
Wheatstone Bridge Circuit Using S.E. in Electronics

The discussion of SE using the example of the price of apples and mangos was perhaps too simplistic to rightfully call a "system," as the price of apples rarely has much to do with the price of mangos. In general, there must be some common situation which the equations describe. In other words, each equation represents one relationship in a system of inter-dependent variables. For instance, the unknowns may be various currents in an electric circuit, and each equation can be a "view" of only one portion of the circuit. Together, the array of numbers generated completely specifies the entire circuit; solving the SE array with MDP SE program cranks out the answers no matter how complex the overall circuit is.

Let's examine a simple circuit which consists of only a voltage source and four resistances, but can be surprisingly tricky to solve intuitively or

using only basic circuit analysis techniques. Electronics buffs will recognize this as the "Wheatstone Bridge" circuit, one used for a long time for comparing resistor values. Usually one resistor would be variable and calibrated, one would be the unknown, and the other two would be fixed of known accuracy. R5 represents the internal resistance of the galvanometer (a sensitive ammeter, or current measuring device). The variable resistor is adjusted until the galvanometer reads zero; in this special case the divider ratio of each resistor branch is the same, from which the unknown value can be computed. But when the bridge is unbalanced, it is not an easy chore to calculate the current I_x, which flows through the meter, or the other currents and voltages involved for that matter. So let's systematically describe the system in an array, feed it to the T/S, and see what comes out.

General Wheatstone Bridge Circuit



Current through galvanometer $I_G = I_3 - I_2$

Current supplied by battery = I_1

NOTE: In this and future problems we assume "conventional" current directions; i.e. current is defined to flow OUT OF THE POSITIVE terminal of voltage sources. This makes it easier to keep signs straight and has the added bonus of making the arrows in semiconductor symbols point the right way.

For a trial run, we assign these values to the resistors:

R1=5
R2=7
R3=6
R4=9
R5=2

(units may be ohms/amps, Kohms/mA., megohms/ μ A. etc.)

Let's start by breaking the circuit into three closed loops, each with its own loop current I_1 , I_2 and I_3 . We assume the currents are all in the same direction, e.g. clockwise; should our assumption prove wrong, the answers we get for I_1 - I_3 will be negative. In this case it is intuitively obvious that the currents actually do flow in this direction, but won't be so in more complex systems. You'll note that R_1 , R_2 , and R_5 appear to have two different currents flowing through them; the actual current that would be measured through these would really be the algebraic sum of the currents, by the principle of superposition. Splitting the circuit into three discrete loop currents just gives us a handle on it, and we can use Kirchhoff's voltage law and Ohm's law to write an equation for each loop. (If these laws aren't familiar, look them up, they're really quite simple and form the basis of a working understanding of electric circuits.) We just have to "walk through" each loop to derive its equation. We put the voltage sources in the loop on the right (the constant) and the resistances on the left (the coefficients). For example, the equation for loop 1 would be:

$$(R_1 + R_2) \cdot I_1 + (-R_1) \cdot I_2 + (-R_2) \cdot I_3 = 10$$

Loop current I_1 flows through R_1 and R_2 , so the first term is $(R_1 + R_2)$. Also flowing through R_1 , however, is loop current I_2 , in the opposite direction to I_1 . The second term is thus $(-R_1)$. Similarly the third term is $(-R_2)$. The equations for the other two loops are:

$$(-R_1) \cdot I_1 + (R_1 + R_3 + R_5) \cdot I_2 + (-R_5) \cdot I_3 = 0 \text{ and}$$

$$(-R_2) \cdot I_1 + (-R_5) \cdot I_2 + (R_2 + R_5 + R_4) \cdot I_3 = 0$$

For our sample run, let's let the resistor values be such that the bridge is obviously unbalanced. Our array would then be:

$$\begin{bmatrix} 12 & -5 & -7 & 10 \\ -5 & 13 & -2 & 0 \\ -7 & -2 & 18 & 0 \end{bmatrix}$$

These values are INPUT, and the program calculates loop currents I_1 to I_3 (stored in $X(M)$, # of rows $M = 3$, # of cols $N = 4$).

So load "SE", select option 5, then choose which method to use. Option 7 runs the problem by Gauss-Seidel and Gauss elimination and does an ill-conditioning check, resulting in the following output:

(Using option 7)

GAUSS SEIDEL CONVERGED
IN 26 ITERATIONS, RESULTS ARE:

X 1	1.5003262
X 2	0.67840835
X 3	0.65883888

RESULTS BY GAUSS ELIMINATION:

X 1	1.5003262
X 2	0.67840835
X 3	0.65883888

ILL CONDITIONING TEST 0.38002366
THE SYSTEM IS OK

(Using option 2)

DENOMINATOR= 1533

RESULTS BY CR:

X 1	1.5003262
X 2	0.67840835
X 3	0.65883888

To use Cramer's rule select option 2 before or after using the other two. To get hard copy press Z when output is displayed; (the program as written works for Timex, Sinclair, and Memotech/Centronics-type printers. For CAI and other systems using USR calls for screen dump, change line 1260 as needed.) Beware of demanding too much accuracy; Gauss-Seidel may tell you the series does not converge when in fact the problem is machine resolution. 1E-8 was used in the example above. Scale your problem if necessary so you have "reasonable" numbers (not very large or very small,) as you lose overall accuracy at either extreme. If your system involves micro-amperes, etc., express your equation in micro-amperes rather than amperes.

Experiment with different values for the above circuit, then work out circuits of your own. When dealing with larger arrays (more loops) be sure you keep the current directions straight. The mutual terms won't always be negative as they are in the simple 2-D circuit above; some resistors will have two or more mutual currents in the same direction. See what happens when you define the same circuit in different ways - define different paths as the

various loops (equations), the results should be the same. Some approaches will be better conditioned, and some may be insoluble. You'll get a good feel of this sort of thing and probably have a great time playing with this or similar circuits.

As long as we use only resistors and fixed voltage sources, any array we derive will show a symmetry along the "minor" diagonals (bottom left to upper right.) Resistors are bi-lateral, which means the mutual terms are the same looking either way. Transistors, FET's, and other neat things can be approximated as linear devices, but with different values for the mutual terms looking either way. In such cases, the coefficient matrix won't simply be a triangular matrix already, simply repeating itself on either side of the major diagonal. Now do you see some of the possibilities? Entire systems can be converted to equations and the solution can be found one way or another.

A comprehensive analysis can still be a lot of work by any definition. You need facility with basic algebra to get the equations into a usable form, and may spend hours arriving at a solution. But compare that with the days or weeks it would have taken with your trusty calculator, a long pencil, and lots of paper. I think you'll agree it's a lot easier to let the T/S and "SE" help out.

Hope y'all have good time with this. I sure did! But now on to brass tacks, super-glue and the like, as I switch hats and give you the first installment of The Custom T/S with some simple reliability mods and an intro to BBU's or "Battery

MATH DEVELOPMENT PROGRAM

Part 2 - Putting it to Work

Ok, Ok, I can hear it now; it's just another simultaneous equation program. Anybody can write one or at least find one in a good computer math book (they seem to be proliferating these days), besides what good are they anyway? I'd now like to present Part 2 in a continuing series; but first, let's hear it for Gauss. I mean really! The man died in 1855 and did for math (and physics) what Newton did for figs (and apples). I don't think pencils had been invented in his time, but he still managed to figure out all of these relationships. Well, enough reminiscence already. The first of three routines up for examination will be "Interpolating Polynomial." This will be followed with Polynomial Regression (Least Squares) and end with a few words on plotting. Along the way we shall touch on a few interesting (I hope) topics which some knowledge of will help demistify this area of math.

Interpolation

This first routine "INTERPOLATES" (as opposed to extrapolate) or calculates values IN BETWEEN known values with the EXACT and only polynomial of the given (Nth) degree that fits the given (N+1) points. The polynomial is unique and independent of the method used to find the equation. This means that regardless of which method you use, the answer should be the same. Beware of roundoff error however. If you compare the results of the SE routines, you can see minor differences from time to time. Since the Interpolating Polynomial deals with definite values (must be a function of X), it will find an exact solution (not withstanding roundoff error). A check on ill-conditioning may show a very ill-conditioned system of equations. This is due in part to manipulating both large and small numbers in the same system creating a truncation error as well as roundoff error.

As the section on fitting goes, I suppose we should start at square one (or should I say, line one); The Straight Line Subroutine. You can find one in your TS/ZX manual, or a better one in the first issue of SWN. Alright, so what does it do? You might say it plays the percentages. If $Y = f(X)$ and you would like to know what Y is between two known X values, the equation is:

$$Y(\text{low}) + (X - X(\text{low})) / (X(\text{top}) - X(\text{low})) * (Y(\text{top}) - Y(\text{low}))$$

Or, in other words, the percentage of the X value above X(low) times the range of the Y values. Put X in a loop from X(low) to X(top), and you will have a tabular, linear interpolation printout. Plotting is slightly tougher though. We have incorporated a compacted version of the Straight line subroutine in the main programs's plot routine. This code checks the slope of the line and then steps between the X values for slopes less than one and steps between the Y values for slopes greater than one. Just make sure your values are on scale first.

Linear interpolation is fine for some applications (such as plotting on the coarse TS/ZX pixels), but is usually only accurate for very small distances away from the known point. To get a better approximation of values between two points, it would be nice to fit the points with a higher order function - say a parabola. Since two points define a particular line, we need a little more information for a parabola. How about three points. A unique parabola is defined by 3 points, a cubic by 4 and so on... You now have the power to fit any number of points that you specify. Don't get too carried away, or the results you get will be worthless.

The interpolation subroutine loads the array in A so as to set up the required number of equations for the given number of points. The equations are Polynomial, of course, but please take note of the method used to generate the equations. Yes, in case you hadn't noticed, more loops. One difference you will see is that the array is filled backwards and an arbitrary constant term is used to help facilitate filling the array as well as validate our polynomial. One, is always a nice number for a constant and is used here. This method is a tremendous memory saving method over any other I know of that will generate any size polynomial. For you fanatics out there, the X determinant is called a Vandermonde determinant. There are other methods, of course, that you can use to solve this type of problem.

These methods of interpolation include Lagrange, which solves a determinant; Aitken's, which takes Lagrange for many points and tells you which degree fit, fits the most points; and Newton's forward and backward difference methods (same guy that invented figs), to name a few. Newton's method is a little beyond our scope here, but it is a widely used method for trying to get a handle on error involved in interpolation. Its device is to interpolate in small steps from one or both ends and therefore reduce error. You must know a lot of points first though. This is a good method for plodding through large tables of data which you must fill in accurately inbetween values (eg., log tables). This method is used by monks and masochistic bosses of high degree.

Lagrange Method

I would at this time like to spend a few moments deriving Lagrange's method, since it pertains directly to part one of this series. Let's develop our determinant for a first degree polynomial. This method won't give a function per se, but it will evaluate a table of data to any degree that we care to program for.

Let's start with the equation $Y = MX + B$ (straight line) where Y is the value we're after, and X is the number we will plug and chug with. B is a constant (Y-intercept) and M, of course, is the slope of the line. We'll let $A0 = B$, $A1 = M$ and $Y = P(X)$. We need to know a point before and after X, called X0 and X1. Their Y coordinates respectively

shall be, $f(X_0)$ and $f(X_1)$. We must also define a constant $K = 1$, to initialize our array. As we plod through our calculations, this constant more or less retains a space for our Y-intercept (A_0) in the determinant.

If $Y = MX + B$ then

$P(X) = A_1X + A_0$ therefore:

$P(X) - A_1X - A_0 = 0$

and by the same reasoning,

$f(X_0) - A_1X_0 - A_0 = 0$

$f(X_1) - A_1X_1 - A_0 = 0$

Since it is a straight line, all of the equations have the same slope A_1 , and Y-intercept A_0 . If the points are real (as opposed to contrived), then there is a non-zero determinant of the coefficients, as follows:

$$\begin{bmatrix} P(X) - X - 1 \\ f(X_0) - X_0 - 1 \\ f(X_1) - X_1 - 1 \end{bmatrix} = 0$$

At this point, I refer you to the Part one section on determinants to achieve the following expansion along the first column and solve for $P(X)$:

$P(X) = f(X_0)((X-X_1)/(X_0-X_1)) + f(X_1)((X-X_0)/(X_1-X_0))$

All values are given except $P(X)$, and X should be a loop counter between X_0 and X_1 . The following program calculates all determinant (det) values first, and then it multiplies each det with its associated Y value and adds them up. Higher degree solutions look like this:

Second degree (3 points)

$Det0 = ((X-X_1)(X-X_2))/((X_0-X_1)(X_0-X_2))$
 $Det1 = ((X-X_0)(X-X_2))/((X_1-X_0)(X_1-X_2))$
 $Det2 = ((X-X_0)(X-X_1))/((X_2-X_0)(X_2-X_1))$

$P(X) = f(X_0)*Det0 + f(X_1)*Det1 + f(X_2)*Det2$

There is a conspicuous pattern that develops with the X's. You can interpolate to any degree your little heart desires by following it. What follows is a little program to calculate a sine table by third degree interpolation. Remember that the ZX/TS uses a radian mode, so to compare you must convert actual sine values to degrees. We will cover this program a little more later on when we talk about error.

```

1 REM SINE INTERPOLATION
5 LET X0=0
10 LET FX0=0
15 LET X1=30
20 LET FX1=.5
25 LET X2=60
30 LET FX2=.8660254
35 LET X3=90
40 LET FX3=1
44 SCROLL
45 PRINT "DEG. LAGRANGE" TR
46 SCROLL
50 FOR X=0 TO 90 STEP 10
60 LET DET0=((X-X1)*(X-X2)*(X-X3))/((X0-X1)*(X0-X2)*(X0-X3))

```

```

70 LET DET1=((X-X0)*(X-X2)*(X-X3))/((X1-X0)*(X1-X2)*(X1-X3))
80 LET DET2=((X-X0)*(X-X1)*(X-X3))/((X2-X0)*(X2-X1)*(X2-X3))
90 LET DET3=((X-X0)*(X-X1)*(X-X2))/((X3-X0)*(X3-X1)*(X3-X2))
100 LET PX=FX0*DET0+FX1*DET1+FX2*DET2+FX3*DET3
110 LET Z=SIN (X/180*PI)
120 SCROLL
130 PRINT X;TAB 5;PX;TAB 16;Z
140 NEXT X
150 STOP
160 SAVE "SINE"
170 RUN

```

"Boy! I just used an interpolating polynomial to fit some data points, and got the wierdest plot! It can't possibly be correct?" While an interpolating polynomial gives an exact equation that goes through every point, those points may not, and probably will not, lie at the maximum or minimum values. It is possible to interpolate a group of data of which two points have coordinates of say, (4,5) and (6,9), with a maximum value in between, such that when $X = 5$, Y may equal 100 or even 1,000,000! This is slightly ill-conditioned to say the least. If you interpolate 8 points then you may have as many as 7 max or min values to contend with. Is there a better way? Well, maybe. You shall be the judge and jury.

Least Squares

```

200 REM LEAST SQUARES
210 PRINT " INPUT NO. OF DATA P
OINTS"
220 INPUT N
222 DIM X(N)
224 DIM Y(N)
230 SCROLL
240 PRINT "DEG. X", "Y"
250 FOR I=1 TO N
260 SCROLL
270 INPUT X(I)
280 INPUT Y(I)
290 PRINT I;TAB 4;X(I);Y(I)
300 NEXT I
310 LET A=NOT PI
320 LET E=A
330 LET C=A
340 LET D=A
350 FOR I=1 TO N
360 LET T=X(I)
370 LET TY=Y(I)
380 LET A=A+T
390 LET E=E+T*T
400 LET C=C+TY
410 LET D=D+T*TY
420 NEXT I
430 REM C=SUM Y
440 REM D=SUM Y^2
450 LET CR=N*E-A*A
470 IF NOT CR THEN STOP
480 LET B=(C*E-A*D)/CR
490 LET M=(N*D-C*A)/CR
500 CLS
510 PRINT "THE FUNCTION IS:",,,
520 PRINT "Y=";M;"X + ";B

```

Least Squares method of curve fitting is another widely used method of curve fitting that will give a smooth fit through a lot of data. If your data is approximate or downright inaccurate, then this method is for you. In contrast to an interpolating polynomial, you get a smooth curve that may not touch any of the given values.

Why call it least squares? Let's go back to our main man, Gauss, again. He wanted a method to fit much collected, ambiguous data smoothly, and deduced that squaring all terms would eliminate all negatives that may cause an inverted fit (probably found out the hard way). An inverted fit is one where all of the points are opposite in sign from

your data. Second, call it Least squares because the object is to have the sum of the squares of the differences between the actual values and the fitted values to be as small as possible (Oh Yes! more statistics!). Smooth curve? Fit approximate data? Sounds great, but just what is needed to make this technique work? To start with, a sound, sober analytical mind that can eyeball data and deduce the right equation for the function without all this hulla-balloo. A trifle too much to ask for probably, but thank God for Gauss and ZX/TS's.

Now, I find it necessary to digress (hopefully evolve for some) momentarily, to cover some background to help in general, the understanding of smooth curve fitting, and in particular, the next few paragraphs as we derive a least squares routine. This subject brings shudders and heart murmurs from those who know about them, but for those of you who don't, have no fear as there is nothing to be afraid of. Now, a few words about Partial Derivatives. This won't substitute for a good Calculus 3 class, but it may alleviate possible ambiguities that may arise. The best way I know how to describe them is with a table. Please examine the following:

Function	Partial Derivatives *****	
	With Respect to X	With Respect to Y
1. 3	0	0
2. X	1	0
3. 3X	3	0
4. X**2	2X	0
5. X**3	3X**2	0
6. YX	Y	X
7. YX**2	2YX	X**2
8. X*XY	1+Y	X
9. Y+XY**2	Y**2	1+2XY
10. Ye ^X	XYe ^X	e ^X

Very simply, the derivative of a constant is zero. If you "freeze" a function in time or space (values are not changing), then a variable with respect to another variable is a constant. At this point, the partial is solved like a normal derivative. As a rule of thumb, if you have a function with more than one variable type ($Z = f(X,Y)$), it is easier to work with two partial derivatives (lines) than it is to work with the function as a whole (variable surface in 3-d). This is especially true in computer programming. Visualizing a surface and programming for it are two entirely different things.

So, what do partial derivatives have to do with the price of tea in China? Not much. Computer curve fitting in general does require solving partial differential equations with respect to each "Degree of Freedom" to determine a curve fit. The degrees of freedom are basically the constants that describe the coefficients for the function in question.

If $f(X) = A1X + A0$, then A1 and A0 are the coefficients to be solved for and we have two degrees of freedom. We will proceed and derive a linear least squares, but don't worry, we have our subroutines for handling this stuff already, thank Gauss!

What is needed for us to make our Least Squares fit? Well, at first glance, we want the fit line as close as possible to all of our points, or, the calculated Y values minus the actual Y values equal to zero. These values are called the Residuals.

Least Squares criterion is actually all of the residual terms squared and added up. You may ask if there is an equation for this, but be patient, it's coming. We have one more term to cover before we proceed, this being the signum or summation notation. This means the sum of all the following terms from a starting value to stopping value say, $i = 1$ to N , but it sounds like grounds for a loop to me. You will see this written in text books as follows:

$$\sum_{i=1}^N$$

N in this expression will be the number of data points we provide.

Let's finally get back to the mainstream and our fundamental equation, $Y = MX + B$, or in our case $f(X) = A1X + A0$. The least square equation for a linear fit is:

$$R = \sum (Y(\text{calc.}) - Y(i))$$

Since $Y(\text{calc.}) = f(X) = A1X + A0$, a direct substitution may be made into the above equation for R (sum of the residuals). We now set the partials of R with respect to each constant to zero and solve two linear simultaneous equations (easy since we're all experts now). The partials of R with respect to:

$$A0 = \sum (A0 + A1X - Y(i)) = \sum 2(A0 + A1X - Y(i)) = 0$$

$$A1 = \sum (A0 + A1X - Y(i)) = \sum 2X(A0 + A1X - Y(i)) = 0$$

The 2's are in all terms and therefore drop out. Multiplying the second equation through by X and moving the Y(i)'s to the other side, gives the following system of equations:

$$\sum A0 + \sum A1X = \sum Y(i)$$

$$\sum A0X + \sum A1X = \sum XY(i)$$

A0 and A1 are coefficients to be determined by the computer and are initialized to one (that marvelous constant giving a helping hand once again). This leaves a simpler system still with which we must manage:

$$\sum 1 + \sum X = \sum Y$$

$$\sum X + \sum X = \sum XY$$

Since we are adding up the values for each X and Y term, the Z1 term becomes the number of data points that are input. This system gives us a good use for Cramer's rule; Least Squares Linear Regression. Now the answer for that question, what do we need for a least square fit?

1. The number of data points.
2. The sum of the X terms.
3. The sum of the Y terms.
4. The sum of the X*Y terms.
5. The sum of the X**2 terms.
6. One Cramer's rule subroutine.

This system assumes that the X values are more accurate than the Y values. If they're not, then swap your X and Y terms to improve your fit. This method of fitting may be used to fit many different types of equations. Logarithmic, Exponential and Power fit methods all use this system of equations to find their solutions. What's the difference? How

the data is manipulated before you sum your variables. These methods simply take the natural log (ln) of the X or Y or both before they are summed. REMEMBER!! The LN of Zero and Negative numbers are undefined and not allowed (program will stop). This leaves only one linear method, to fit data with values less than or equal to zero. Keep in mind that there are as many ways to fit data as there are pixels on the screen, so don't let a bad fit using one method stop you from trying some other method (memory permitting). With a little knowledge of statistics and enough degrees of freedom you can fit your data. It just requires a function and partial derivatives for each coefficient (no fun, but true none-the-less).

If your data appears to resemble a polynomial of some degree, then you might try our nifty little polynomial regression routine. Actually, it is the same routine as the linear regression, but uses the interpolation routine to derive a function. You may derive as high a degree of fit as your memory dictates. the routine creates an array (C) to fill array A, and then deletes array C when it is finished. Since not all of you have extravagant amounts of memory to play with, I have a more conventional linear least squares routine following this. It does the same thing as the one in the main program, but is easier to follow. Most programs on the market use this approach. Try cubic regression and see how memory efficient it is (needs Gauss elim. to solve).

Polynomial Interpolation-Regression

At this point, I would like to recap briefly what we have covered. First, the simultaneous equations and their derivation. Second, interpolation in general, with a derivation of the LaGrange polynomial and a few words about the straight line subroutine. Third, Least Squares curve fit with a linear derivation and a few words about partial derivatives.

These routines are fairly easy to derive and implement, but what about higher orders of magnitude? How do the equations compare? What does the computer see? Are the results accurate? Even better, will your boss believe you if you tell him you did it on a ZX/TS rather than an IBM mainframe?

First, my opinion on the last question; if you boss starches his collar, then don't tell him right away that a TS1000 solved his problem. Show him the good work first. As for the other questions, let's compare quadratic interpolation and quadratic regression and look at what the computer sees. The two routines, Interpolating Polynomial and Polynomial Regression are not easy to follow (just try some other way to get all that in 16K), albeit there are relatively few calculation lines. Triple loops have a habit of concealing vast amounts of computation in a few (even one) calculation lines. Interpolation uses a double loop for initializing Y values and setting all X values to one, a triple loop to build the X determinant into its higher order (backwards, from right to left) and a single loop to switch the coefficients back around on returning from the Gauss elimination subroutine.

Along the same lines, the regression routine uses a single loop to initialize column one of array C to one, a double loop to load and build array C with X values, a triple loop to load array A with array C values and further build array A and a double loop to load the last column of A with the sums of the Y values * the first column of X values from array C. (Remember, column 1 of C started with all ones.)

Here are the equations and systems we are solving for:

QUADRATIC INTERPOLATING POLYNOMIAL

$$X(1)**2 + X(1) + 1 = Y(1)$$

$$X(2)**2 + X(2) + 1 = Y(2)$$

$$X(3)**2 + X(3) + 1 = Y(3)$$

QUADRATIC REGRESSION POLYNOMIAL

$$\sum N + \sum X + \sum X**2 = \sum Y$$

$$\sum X + \sum X**2 + \sum X**3 = \sum X*Y$$

$$\sum X**2 + \sum X**3 + \sum X**4 = \sum X**2*Y$$

As you can see, the equations are quite polynomial, and follow a clear pattern of development. This makes any degree polynomial relatively easy to program (memory permitting).

The difference between linear and quadratic interpolation is column 1 and row 3; for regression is column 3 and row 3. Remove these rows and columns from the system and you have the linear form. The computer sees only array A in its computation (dynamically dimensioned as needed as DIM A(M,N)), necessitating a lot of loops to fill it (loops keep the program length down).

Error, on the other hand, is not so easy to get a handle on. Fortunately, Least Squares negates the effect of error by running an average line through all of the data points, acknowledging that every data point may be inaccurate. The residuals should equal zero (the sum of the f(X)-Y values, also the X*(f(X)-Y) values. These statistics may be checked after F\$ has been calculated in the program. Linear regression can be checked for goodness of fit by using the Pearson Product Moment Correlation Coefficient, easily and efficiently by adding up all the Y squared terms as well (Y2). A perfect fit =1; good =>.8. This term is not of much value for higher orders of fit.

Interpolation is supposed to give an exact equation, but I'm sure you all know better by now. (I hope so anyway.) Look to the N+1 derivative of the function for the error solution. If you have a third degree polynomial, then look at the fourth derivative. Say WHAT!! Did I hear you say that the fourth derivative of a third degree polynomial is ZERO? You whizzes are really on the ball out there. It is an EXACT polynomial, right? Well, let's take another look at the sine interpolation demo. The

fourth derivative of sine is still sine, and it is not zero. Because the computer calculates in radian mode (1 Radian = 57.29 degrees), the conversion to meaningful numbers is made. Examine the following terms:

$$F(X) = \text{SINE}(X/57.29)$$

$$F'(X) = \text{COS}(X/57.29)/57.29$$

$$F''(X) = -\text{SINE}(X/57.29)/57.29^2$$

$$F'''(X) = -\text{COS}(X/57.29)/57.29^3$$

$$F''''(X) = \text{SINE}(X/57.29)/57.29^4$$

If the maximum value for sine is one (90 degrees), then $1/57.29^{**4}$ is the largest term that the fourth derivative can equal. To determine the maximum error, determine the difference between all known data points and the value at which you want to determine the error at. Multiply that by the max error and divide by N factorial (N=power of the derivative you are looking at; 4 in this case). For the sine of 10 degrees, the maximum error can be no larger than:

$$1/57.29^{**4} * (10 - 20 - 50 - 80)/(4 * 3 * 2 * 1) = -3.1E-3$$

The accuracy is better than two decimal places at worst. This calculation is comparable to the remainder term of a Taylor series (another bag of worms). When you reach a given value of X, then a

zero enters the equation and the error term of course becomes zero and drops out (watch out for roundoff error. Since we know a few of the terms (even smaller than 1), then we know that the error is less. This also indicates that the error is larger as we approach 90 degrees. Have fun with this one!



'SE' (continued from Part 1)

```

4000 REM PRINT SCREEN
4010 PRINT "WOULD YOU LIKE TO PR
INT FROM: ", "0. INPUT DATA", "1
. CURVE FIT", "2. OTHER FUNCTION
4020 SLOW
4030 INPUT Z
4040 GOTO (4050 AND NOT Z)+(4140
AND Z=1)+(4120 AND Z=2)
4050 GOSUB 910
4060 PRINT "0. ", "F."
4070 FOR I=1 TO DP
4080 GOSUB 910
4090 PRINT I;TAB 4;P(I),Y(I)
4100 NEXT I
4110 GOTO 4300
4120 PRINT " INPUT F(X)"
4130 INPUT F#
4140 PRINT " INPUT X START";
4150 INPUT XS
4160 PRINT XS
4170 PRINT " INPUT X STOP ";
4180 INPUT XP
4190 PRINT XP
4200 PRINT " INPUT STEP VAL "
4210 INPUT INC
4220 CLS
4230 IF LS OR Z THEN PRINT AT 16
,D;"THE FUNCTION IS:",F#
4240 GOSUB 910
4250 PRINT "0. ", "F."
4260 FOR X=XS TO XP STEP INC
4270 GOSUB 910
4280 PRINT X,VAL F#
4290 NEXT X
4300 GOSUB 910
4309 POKE 16418,0
4310 PRINT AT 21,30;" INPUT ANY
KEY-""Z"" TO COPY

```

;Print menu, if you would like to LPRINT then insert a line in the right place. There is plenty of room.

;input any function here. Go back and scale it and plot it to see what it looks like.

;This is a common return point for these later routines

```

4320 PAUSE 4E4
4330 FAST
4340 IF INKEY$="Z" THEN COPY
4350 IF LS THEN RETURN
4360 LET M=DP
4370 LET AL=Z0
4380 LET LS=Z0
4390 RETURN
4400 REM ENTERED LINE 4400
4410 PRINT " INPUT POSITION, IN
ORDER OF", "ENTRY, OF THE FIRST X
TERM TO BEEVALUATED"
4420 INPUT POS
4430 PRINT "DO YOU WANT TO FIT
THE REST OF THE POINTS? YES"
4440 INPUT AL
4450 IF AL THEN PRINT "YOU CANN
OT PLOT DIRECTLY.", "WANT A LIST
?YES"
4460 IF NOT AL THEN GOTO 4500
4470 INPUT LS
4480 PRINT " INPUT STEP VAL "
4490 INPUT INC
4500 FAST
4510 LET POS=POS-A
4520 LET LAST=POS+DEG
4530 IF LAST>DP THEN LET DEG=DP-
POS
4540 LET M=DEG
4550 LET IP=PI
4560 LET N=M+A
4570 DIM A(M,N)
4580 FOR I=A TO M
4590 LET A(I,N)=Y(I+POS)
4600 FOR J=A TO M
4610 LET A(I,J)=A
4620 NEXT J
4630 NEXT I
4640 FOR J=M-A TO A STEP -A
4650 LET DE=DE+A
4660 FOR I=A TO M
4670 FOR K=A TO DE
4680 LET A(I,J)=A(I,J)+P(I+POS)
4690 NEXT K
4700 NEXT I
4710 NEXT J
4720 LET DE=Z0
4730 GOSUB 1750
4740 LET IP=Z0
4750 LET L=M
4760 FOR I=A TO M/TW
4770 LET T=X(I)
4780 LET X(I)=X(L)
4790 LET X(L)=T
4800 LET L=L-A
4810 NEXT I
4820 LET F$=""
4830 FOR I=A TO M
4840 PRINT "X"; I; " = "; X(I)
4850 LET F$=F$+STR$ X(I)+"*X**"+
STR$ (I-1)+" "
4860 IF I<>M THEN LET F$=F$+"+"
4870 NEXT I
4880 IF NOT AL THEN GOTO 5486
4890 LET XS=P(POS+A)
4900 LET XP=P(LAST)
4910 SLOW
4920 GOSUB 4230
4930 LET POS=LAST
4940 IF POS=DP THEN GOTO 4360
4950 CLS
4960 GOTO 4500
4980 REM ENTERED LINE 4980
4990 CLS
5000 FAST
5010 FOR I=A TO M
5020 IF Z=A OR Z=TW OR Z=A+D THE
N LET U(I)=P(I)
5030 IF Z=TR OR Z=D AND P(I)>Z0
THEN LET U(I)=LN P(I)
5040 IF Z=A OR Z=TR OR Z=A+D THE
N LET V(I)=Y(I)
5050 IF Z=TW OR Z=D AND Y(I)>Z0
THEN LET V(I)=LN Y(I)
5060 NEXT I
5080 REM ENTERED LINE 5080
5090 IF Z AND Z<A+D THEN LET DEG
=TW
5100 LET M=DEG
5110 LET N=M+A
5120 DIM A(M,N)
5130 DIM C(DP,M)
5140 LET Y2=Z0
5150 FOR I=A TO DP
5160 LET C(I,A)=A
5170 LET Y2=Y2+V(I)*V(I)
5180 NEXT I

```

;If you want to start with X(2) then input 2. This type of routine is only valid for the middle values. POS=start POSITION in your data. It will take the number of points that you specify in the menu at line 690 or ALL of them and give a list (LS) in increments of INC. If you have an odd # of points then the last few points will be fit by the degree most fitting (4510--4530). I hope that makes it perfectly clear!

;IP= Interpol. Poly. flag
;Starts here

;Set Y values

;Set determinant = 1

;Fill array backwards

;Build array

;Come back and flip values around

;This is the function table for building a polynomial in F\$. It isn't pretty but it works!
;You should save or Lprint F\$ if you are interpolating a lot of data. It is not saved anywhere.

;Go back for more points.

;This table sets up the least squares according to your choice from line 690. It assigns LOG values to the fit array if necessary.

;This starts that nifty regression routine. It will fit any degree that you specify and have the memory for.

;Y2 is for the correlation coefficient

;Initialize C array
;Sum Y2

```

5190 FOR J=TW TO M
5200 FOR I=A TO DP
5210 LET C(I,J)=C(I,J-A)*U(I)
5220 NEXT I
5230 NEXT J
5240 FOR I=A TO M
5250 FOR J=A TO M
5260 FOR K=A TO DP
5270 LET A(I,J)=A(I,J)+C(K,I)*C(
K,J)
5280 NEXT K
5290 NEXT J
5300 NEXT I
5310 FOR I=A TO M
5320 FOR J=A TO DP
5330 LET A(I,N)=A(I,N)+C(J,I)*V(
J)
5340 NEXT J
5350 NEXT I
5360 DIM C(A)
5370 REM *****
5380 LET R=A(TW,N)-(A(TW,A)*A(A,
N))/DP
5390 LET R=R/R
5400 LET R=R/(((A(TW,TW)-(A(TW,A)
)*A(TW,A))/DP)+(Y2-(A(A,N)*A(A,N)
))/DP)))
5410 LET IP=A
5420 GOSUB (2650 AND M=TW)+(1750
AND M>TW)
5430 LET IP=Z0
5440 IF M>TW THEN GOTO 4820
5450 LET F$=""
5460 IF Z=A THEN LET F$=STR$ X(A
)+"X"+STR$ X(TW)
5470 IF Z=TW THEN LET F$=STR$ (E
XP X(A))+"*EXP (X"+STR$ X(TW)+"
)"
5480 IF Z=TR THEN LET F$=STR$ X(
A)+"*LN X"+STR$ X(TW)
5485 IF Z=D THEN LET F$=STR$ (EX
P X(A))+"*X"+STR$ X(TW)
5486 REM *****
5487 LET RXY=Z0
5488 LET RY=Z0
5489 FOR I=1 TO DP
5490 LET X=P(I)
5491 LET Y=Y(I)
5492 LET T=VAL F$
5495 LET RY=RY+(T-Y)*(T-Y)
5496 LET RXY=RXY+X*(T-Y)*(T-Y)
5498 NEXT I
5500 CLS
5510 PRINT "THE FUNCTION IS:",F$
5520 IF Z THEN PRINT "CORRELATIO
N COEFFICIENT:",R
5522 IF Z THEN PRINT "RESIDUAL
S","Y*S",RY,"XY*S",RXY
5530 GOTO 4309
5540 REM *****
5550 PRINT " INPUT START VAL FOR
X FIT"
5560 INPUT XS
5570 PRINT " INPUT STOP VAL FOR
X FIT"
5580 INPUT XP
5590 PRINT " INPUT CURVE FIT P
OINT DENSITY"
5600 INPUT PD
5610 FAST
5620 LET K=LN 10
5630 GOSUB 146
5640 IF NOT XL THEN LET TXE=E(A)
5650 IF NOT YL THEN LET TYE=E(TR)
5660 IF XL THEN LET TXE=LN E(A)/
K
5670 IF YL THEN LET TYE=LN E(TR)
/K
5680 LET PF=A
5690 DIM F(PD+A)
5700 DIM G(PD+A)
5710 LET DF=(XP-XS)/PD
5720 FOR X=XS TO XP STEP DF
5730 LET FX=VAL F$
5740 IF FX<E(TR) OR FX>E(D) THEN
GOTO 5860
5750 IF NOT XL THEN LET TX=X
5760 IF XL THEN LET TX=LN X/K
5770 IF YL THEN LET FX=LN FX/K
5780 LET TX=(TX-TXE)/DX
5790 LET FX=(FX-TYE)/DY
5800 LET TX=TX+7.5
5810 LET FX=FX+3.5
5820 IF TX>64 OR FX>44 THEN GO
TO 5860
5830 LET F(PF)=INT TX
5840 LET G(PF)=INT FX
5850 LET PF=PF+A
5860 NEXT X
5880 SLOW
5890 RETURN

```

;Build C array

;put C in A array

;Build Y values

;How good a fit is it? This is only really valid for linear regression (choices 1-4)

;This is the linear regression function table.

;Residuals will give a good indication of fit.
;Closer to zero the better the fit.

;Pick the range you want to fit. This is where an auto fit routine would come in handy. A short routine to preassign these values at line 800 would be very nice (maybe take the first and last values initially)
;40 points is a good density for a ZX81
;For 2068--a good project would be to fit a curve, THEN use an interpolating poly. to give DRAW a third parameter. You may then connect your points with a curve by DRAWing.

;Set fit = graph scale

;Calculate fit
;if it is not on scale then skip it.

;F and g are fit curve plotters.

MATH DEVELOPMENT PROGRAM

Part 3 Integration

Integration has been quite a civil subject in the last twenty years. Fortunately for us, our predecessors, Gauss, Simpson, Newton, etc., solved our transcendental integration problems and developed formulas for calculating the integral of a function. These old methods are still used rather extensively today. There have been some refinements to some of these techniques, to expedite number crunching both by hand and on a computer. You can find a short routine in most computer texts that deal with solving some function (you must know in advance) by one method or another, and spit out the area under the curve. What if you just have data points? How accurate will your results be? What if you have four or five variables and want to blanket a range of values for each variable (might take all day with some methods)? Wouldn't it be nice to compare several methods without taking any more time than finding the answer with one method? Well, please stay tuned.

If you have ever taken a calculus course, then you are quite familiar with the various techniques of integration. You may recall learning to first differentiate and then integrate, but since computers can integrate so easy, we'll start the other way around. To jog your memories, aside from straight forward polynomial integration there is parts, trig. substitution, partial fractions and a few other nasties that are taught, not to mention the hundreds of integrals found in the tables. Fortunately, all those text book problems have text book answers. However in real life, those arbitrary shapes may have no "closed form" expression. In other words, don't bother looking in the integration tables. When faced with this type of problem, what do you do? You must either break your problem into known pieces and evaluate them separately or resort to some numerical technique to treat it. Numerical methods for solving integrals are called Quadrature. The methods we will touch on are Interpolation (of course), which is a clarification term for Newton-Cotes (general form for Trapezoidal, Simpson and Romberg rules) and the method devised by our ally, Gauss; Gauss Quadrature.

One problem I pondered was how much derivation and detail to go into in describing how and what these routines can do. Considering all the books out on the market dealing with just this subject, whenever you see "gory details", think of about 4 pages of nasty formulas deriving equations. In order to condense this subject, some tables have been given with relevant data to spare your having to dig them up from somewhere. On the bright side though, quadrature listings are rather short and fairly easy to implement.

Now, just what is integration? Don't bother looking in the dictionary. It says, "The operation of finding a function whose differential is known". Those guys are really on the ball, aren't they? This is true but it won't help us much. Numerically speaking, how about the operation of finding the area or volume defined by given data or a function, the X axis (for now), and some start (L) and stop (H) limits (to define DX with).

Trapezoidal Method

The area of a square or rectangle is the height * base (side * side); a triangle, $1/2 * \text{height} * \text{base}$. Simple right? Do you remember the area of a trapezoid? A slight variation of the triangle and/or rectangle. A trapezoid has 2 parallel sides and 2 sides slanted from each other. Add the two parallel sides together as the height and treat it like a triangle, or make the smallest rectangle that will contain the trapezoid and add it to the area of the largest one that will fit inside the trapezoid and divide by two. You get the same result either way. The Trapezoidal rule follows:

$$\text{AREA} = \text{DX}/2 * (\text{f}(\text{start}(\text{L})) + \text{f}(\text{stop}(\text{H})))$$

Least Squares

Using linear regression (least squares), you need only evaluate the first and last points (trapezoid) and divide by two to find the area. In the SE program, that includes choices 1-4 of the plot routine and first degree interpolation.

Simpson's Rule

What about a curved line? One method would be to divide the base (DX) into smaller and smaller sections, first into 2, then 4, 8, 16, 32, 64, 128, ... all the way to infinity. Text book solutions do just this to arrive at the exact answer. This works for quadrature only to a certain point. Why? Round off and truncation error, not to mention infinity calculations! 128 trapezoids will probably give the most reasonable results before error starts to take you away. A better method for curved shapes would be Simpson's rule (sparing the gory details). This method accurately defines a parabola and actually up to a cubic function by the following formula:

$$\text{AREA} = ((\text{DX}/3) * (\text{f}(\text{start}(\text{L})) + 4 * \text{f}(\text{mid}(\text{M})) + \text{f}(\text{last}(\text{H}))))$$

Romberg Integration

Romberg came along later on and noticed a relationship between the various methods that were discovered somewhat independently at different times in history. By doing some number finagling (no gory details), Romberg found that he could get the accuracy of 128 trapezoids by evaluating only 16! He noticed the following relationship and tied (mathematically) the rules together.

If we let T1 = the area determined by one trapezoid, T2 = the area by two trapezoids and S1 = the area by using Simpson's rule one time (two areas evaluated slightly differently from two trapezoid areas), he found the following:

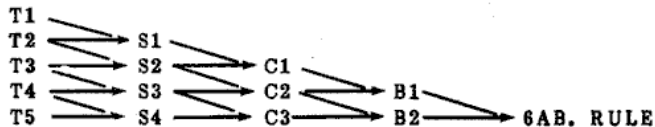
$$S1 = (4 \cdot T2 - T1)/3 \text{ where}$$

$$T1 = DX/2 \cdot (f(B) + f(H))$$

$$T2 = DX/4 \cdot (f(B) + 2f(M) + f(H))$$

$$S1 = DX/2 \cdot (f(B) + 4f(M) + f(H))$$

It turns out that Rombergs rule refines values INTERPOLATED by trapezoid boundaries. The more trapezoids, the higher the interpolation, as follows (see the second table for further clarification):



The error decreases by a factor of 4 for every step down col. 1, and $4^{(N-1)}$ as you go across (N = col no.). $S1$ is 16 times more accurate than $T1$. $B1$ however has 6 decimal places better accuracy, and the 6 abscissa rule is better than 10 places, which is as good as you can do before truncation error takes over. $B2$ at 8 place accuracy should also give an exact answer. This system has some drawbacks that you should be aware of. First, your points must be spaced evenly and you need the right number of points in multiples + 1 of the degree you desire. For Simpsons rule (second degree), you need an odd number of points. For Cotes (third degree), you need 4, 7, 10, etc., points. This method is slow, but it is much faster than evaluating 128 trapezoids. For four iterations of Romberg, you need to evaluate 17 data points ($2^{(N+1)}$). The first three iterations take about one to two seconds apiece to evaluate, the fourth takes about four. The Romberg correction to 128 trapezoids takes about another two seconds. 128 trapezoids goes into several minutes to evaluate when summed independently.

If your points are unevenly spaced then interpolate it with SE and use VAL F\$. The answer should be the same. Since integrating a particular function or set of data can be very specific and hard to generalize about, what follows is a table to pick specific equations from to help speed up your basic programming.

OK, now you've got several methods for evaluating polynomials, but the fastest way to get better than 6 decimal place accuracy requires about 10 seconds. A polynomial function can be integrated by sight or by hand that fast (if it's an easy one). What if your equation is $1/X^{**2}$, and you need to integrate between 0 and 2? All the methods described fail. What if your equation has 5 variables and you need them all tested between 0 and 10? Oh, one more thing; you need all the answers in an hour. Well, don't skip town because Gauss has done it again with:

Gauss Quadrature

This technique is the fastest and what I consider the best all around integration method. Up front though, there are disadvantages to this method. Unless you have a real time clock, discrete data points can't take advantage of this method. You need a function. A dimension statement is required to hold the coefficients and weights, and the method is only good with functions whose limits are between -1 and +1. It is also very difficult to tell how much error is involved in your answer. As you take more points, you don't approach your answer, instead you hit all around it (you may also hit right on it but not know it). This appears to severely limit the usefulness of Gauss Quadrature (GQ). In fact, GQ is the most widely used method because of its speed. We have advanced function development capabilities at our beck and call. We have room for a dimension statement, and now all we need is a method to transform any function to begin at -1 and end at +1.

NEWTON-COTES FORMULAS FOR INTERPOLATING POLYNOMIALS

Romberg Column	Rule Used	Polynomial Equivalent
ONE	Trapezoidal (given above)	$Y = A1X + A0$
TWO	Simpson (given above)	$Y = A2X^{**2} + A1X + A0$
THREE	Cotes $3DX/8 \cdot (f(L) + 3 \cdot f(1/3) + 3 \cdot f(2/3) + f(H))$	$Y = A3X^{**3} + A2X^{**2} + A1X + A0$
FOUR	Booles $2 \cdot DX/45 \cdot (7 \cdot f(L) + 32 \cdot f(1/4) + 12 \cdot f(1/2) + 32 \cdot f(3/4) + 7 \cdot f(H))$	FOURTH DEGREE
FIVE	6 Abcissa (point) Rule $5 \cdot DX/288 \cdot (19 \cdot f(L) + 75 \cdot f(1/5) + 50 \cdot f(2/5) + 50 \cdot f(3/5) + 75 \cdot f(4/5) + 19 \cdot f(H))$	FIFTH DEGREE

Reasons for using GQ are the ability to integrate to infinite limits (provided the first point tested < 1e38) and SPEED. You can run through vast combinations variables, $A*X**2+B*X+C$, by putting A, B, C, and X in loops and calculating 3 or 4 integrals per second! In BASIC!

Now, the method to this madness (again no gory details). Using trapezoids, you need the first and last points. With Simpsons rule, you are required to have the mid-point as well. Gauss wanted a general way to evaluate some tough functions without going through 500 trapezoids to get better accuracy. Using simultaneous equations, he proved that as long as the limits were +/-1, any function evaluated where $X = \pm(1/\sqrt{3})$, added up and divided by 2 would equal the area under the curve (up to third degree). That means evaluating two points to determine a cubic polynomial!

Implementing GQ is simple and straight forward. When evaluating from low (L) to high (H), $DX = (H-L)/2$ and each X term evaluated must be converted to a relative position between +/-1 as if L = -1 and H = +1. This simple conversion is:

$$X' = ((H-L)*X+H+L)/2$$

X, in this equation for two points, = (+/-) $1/\sqrt{3}$ and X' now becomes the new X. After this, evaluate F\$, multiply by the Y weighting factor, add them up and multiply by DX and there you have it. (practically painless, isn't it?) By the way, this is all done in TWO calculation lines. Another table of values is given to save you the trouble of looking for them. The program as listed, contains the coefficients and weights for 2, 3 and 4 points. The numbers look strange but are in accordance with the convention used in the SE program and can be incorporated quite easily. Add a 7. INTEGRATION at the main menu and at the plot menu with the appropriate GOSUB and RETURN. Believe it or not, but

this simple routine is used on those big mainframe computers in analysis of bridges, aircraft, etc. Remember, Static (Strength of Materials) and Dynamic theory for structure evaluation evolved about the time of Gauss' departure. Before this, those bridges collapsed a lot more often.

GAUSS POINTS and WEIGHTS

DATA POINTS	X POINTS	Y WEIGHTS
2	+/- .57735027	1.0
3	0.0 +/- .77459667	.88888889 .55555556
4	+/- .33998104 +/- .86113631	.65214515 .34785486
5	0.0 +/- .53846931 +/- .90617985	.56888889 .47862867 .23692689
6	+/- .23861919 +/- .66120939 +/- .93246951	.46791393 .36076157 .17132449

There are other methods used for integrating, including Monte Carlo, which generates RANDOM numbers that either hit (below VAL F\$) or miss (above VAL F\$). This method is very slow. For 3 place accuracy, you need at least 4 orders of magnitude (10,000) evaluations of RAND and VAL F\$. Leave this slow poke for the really fast mainframes. The text is short this time, but the routines are packed. Some good examples will be forthcoming, but I believe you can find lots of good applications for these routines. Next time, I will expose some Differentiation techniques that I hope you will find useful.

'SE' (continued from Part 2)

```

5997 REM INTEGRATE
5998 PRINT "1) GAUSS QUAD", "2
) SIMP.-DATA", "3) SIMP.-FUNC.",
"4) ROMBERG"
6000 INPUT Z
6001 FAST
6002 GOSUB (6060 AND Z=A)+(6500
AND Z=TW)+(6600 AND Z=TR)+(7200
AND Z=D)
6003 PAUSE 4E4
6004 GOTO 1000
6005 PRINT " INPUT FUNCTION? 1/0
NO=0"
6006 INPUT X
6007 IF X THEN INPUT F$
6010 PRINT "INTEGRATION", " INPU
T START THEN STOP LIMITS"
6020 INPUT L
6030 PRINT L,
6040 INPUT H
6050 PRINT H
6051 RETURN
6060 REM GAUSS QUAD
6070 GOSUB 6005
6080 DIM S(TR,D)
6090 DIM T(TR,D)
6100 LET S(A,A)=A
6102 LET S(A,TW)=A
6104 LET S(TW,A)=(A+D)/(TR*TR)
6106 LET S(TW,TW)=(D+D)/(TR*TR)

```

;if you need to input a new function then input 1 first.
;input 0 and retain old F\$

;Input Low and High Limits

;This routine has data for Gauss Quadrature of 2, 3 and 4 points. S(3,4), T(3,4). For 6 points, DIM S(5,6), T(5,6) with Y weights in S and X points in T

```

6100 LET S(TW,TR)=S(TW,A)
6110 LET S(TR,A)=.65214516
6112 LET S(TR,TR)=S(TR,A)
6114 LET S(TR,TW)=.34785485
6116 LET S(TR,D)=S(TR,TW)
6120 LET T(A,TW)=A/SQR TR
6122 LET T(A,A)=-T(A,TW)
6124 LET T(TW,TR)=SQR (TR/(D+A))
6126 LET T(TW,A)=-T(TW,TR)
6128 LET T(TR,TR)=.33998104
6130 LET T(TR,A)=-T(TR,TR)
6132 LET T(TR,D)=-.86113631
6134 LET T(TR,TW)=-T(TR,D)
6135 PRINT " INPUT NO. OF POINTS
2(<=)4"
6136 INPUT DEG
6140 LET DX=(H-L)/TW
6142 LET FX=Z0
6150 LET K=DEG-A
6160 FOR I=A TO DEG
6180 LET X=((H-L)*T(K,I)+H+L)/TW
6190 LET FX=FX+S(K,I)*VAL F$
6210 NEXT I
6220 LET FX=FX+DX
6230 PRINT "GAUSS QUAD= ",FX
6250 RETURN
6501 REM GAUSS QUAD
6510 LET FX=Z0
6520 LET DX=U(TW)-U(A)
6550 FOR I=A TO DP-TW STEP TW
6560 LET FX=FX+DX/TR*(Y(I)+D*Y(I
+A)+Y(I+TW))
6570 NEXT I
6590 PRINT "SIMPSONS AREA = ",FX
6599 RETURN
6600 REM SIMPSONS AREA
6610 GOSUB 6805
6630 PRINT " INPUT MAX ITERATION
S. (<=8)"
6635 INPUT IT
6640 CLS
6650 FOR I=1 TO IT
6660 LET DIV=2+I
6670 LET AR=Z0
6680 LET DX=(H-L)/DIV
6683 LET T=DX/TR
6690 LET SIM=DIV/TW
6700 FOR J=A TO SIM
6705 LET FX=Z0
6720 LET X=L+TW*(J-A)*DX
6725 LET FX=VAL F$
6730 LET X=X+DX
6732 LET FX=FX+4*VAL F$
6735 LET X=X+DX
6740 LET AR=AR+(FX+VAL F$)*T
6810 NEXT J
6820 PRINT " FOR ";DIV;" DIVISIO
NS, AREA =",AR,,
6830 PAUSE 50
6835 IF Z>TR THEN RETURN
6840 NEXT I
6900 RETURN
7200 REM ROMBERG TRIANGLE
7210 GOSUB 6805
7300 PRINT " INPUT MAX ITERATION
S. (<=8)"
7310 INPUT IT
7320 DIM A(IT,IT)
7325 CLS
7330 FOR I=A TO IT
7340 LET DIV=TW*(I-A)
7350 LET AR=Z0
7360 LET DX=(H-L)/DIV
7365 LET T=DX/TW
7370 FOR J=A TO DIV
7375 LET FX=Z0
7380 LET X=L+(J-A)*DX
7390 LET FX=FX+VAL F$
7400 LET X=L+J+DX
7410 LET AR=AR+(FX+VAL F$)*T
7420 NEXT J
7425 LET A(I,A)=AR
7430 GOSUB 6820
7450 NEXT I
7500 REM ROMBERG TRIANGLE
7505 CLS
7530 FOR J=TW TO IT
7535 LET T=D*(J-A)
7540 FOR I=J TO IT
7560 LET A(I,J)=(T+A(I,J-A)-A(I-
A,J-A))/(T-A)
7570 NEXT I
7580 NEXT J
7585 SLOW
7590 FOR I=A TO IT
7595 PRINT
7600 FOR J=A TO I
7610 PRINT A(I,J); " ";
7620 NEXT J
7625 PRINT
7630 NEXT I
7640 FAST
7650 RETURN

```

;These lines maybe deleted after you run and verify your answer. Just don't RUN it again.

;Put DEG in a FOR NEXT loop from 2 to 4 at line 6138 and get Gauss Quad results for three values

;These two lines are the meat of the matter.
;Convert X, EVALuate F\$ and sum them up.

;Multiply X*Y to get area.
;6240 NEXT DEG

;This routine needs an odd number of points to work as is.

;It does all the Simpson calculations in one line
;It will evaluate discrete points in SE just fine.

;This tricky routine requires a function. It evaluates the whole area, divides it in two and evaluates each of them, divides each of them into two and evaluates each of them and so on. Watch the computer approach the answer.

;Look at Fred's intro on this routine for a less complex listing

;This Rule also uses the same technique of dividing the area. Four iterations of this routine followed by Romberg, will give as much accuracy as your computer can handle.
;Functions with X in the denominator may not converge in four iterations. After that, roundoff error may take you away. Beware!

;Try /(X**2+1) for a function. Compare your answer to PI.

;This short routine constitutes the Romberg triangle. This is an excellent method.

How to Use 'SE'

Finally, the last thing on my agenda for this volume is the documentation for the program. Please, read the annotations! There is a lot of information in the listing itself. I know of more than one person who "gritted their teeth" and drank a lot of coffee, and succeeded in entering the listing by hand. They truly wanted to LEARN this fine art. I have not included a flow chart or a listing of variables, however many are defined in the listing. If you make a working copy, then most initialization and quadrature LET statements can be left out. I also did not assign subroutines to variables, for example, GOSUB 1760, could be GOSUB GE, if you LET GE=1760. This would be even more of a savings on space. Making the PLOT arrays into string arrays would create a whopping big space savings. If you have a few hundred data points and you really want to manipulate them (in 16K), then this may be a necessity.

What started out to be a small, quick and dirty method of viewing data, blossomed into a rather fine, option driven plot program (in amongst all the other things this program will do). Plotting and fitting data, applies virtually every aspect of what we have covered here and quite a bit more.

This program makes rather extensive use of arrays (keeps computation loops simple), and you really don't have to worry about losing your data by mistake. If you have a lot of data (more than 70 points) then you may need more than 16K. Basic flags are used to keep track of what is going on in the subroutines, and "Z" is used for screen dumps. Although array plotting is not memory efficient, it is considerably faster than plotting and calculating at the same time. With the ability to recall data or functions, rescale and refit, you have flexibility not typical of many ZX/TS programs. A sample senario follows:

1. LOAD "SE". When the main menu appears, press 6 (to input data for plotting). "How many data points?" will appear. Give a number and then enter X and Y values in pairs. Printing is done after each pair. Next, the menu asks, "Whay would you like to do?" You MUST plot your data first (choice 1). This will set the parameters for scaling. (This is done to save space. You could initialize all these flags to zero and set screen size around line 1000 if you want.) You need never label your graph or give it a name until your ready to COPY. Set your scale (LIN/LOG) and set your limits (LOW/HIGH) for graphing. This is only necessary on the first plot. These values are retained until you change your graph limits. Connect your points at any time (this is a screen function), but only plot a curve fit after you have fit it (2 and 3).

2. Fitting the data comes next (choice 2). Notice on the various screen dumps, just how many ways you can fit the same data points. All of the graphs are of the same set. Remarkable, isn't it? There are so many possibilities here that I couldn't

come close to naming them. You have four linear regression, any size interpolation (each degree is different) and polynomial regression to choose from. Just experiment until you get a good fit (part of the learning curve). Remember, no negative values or zero for any data in linear regression except linear (LIN/LIN on both X and Y axes) itself.

3. Scale your choice of fit if you wish to plot it (choice 3). This routine was separated from the fit routine, to accomodate the interpolation routine shortcomings. They are not really shortcomings, because interpolation is done in small pieces. For example, if you have 10 data points to interpolate, then (look back at the form for interpolating polynomial) there will be a X^{**9} term in every equation. You are losing accuracy at a very fast rate. You are much better off taking them three or four points at a time. Unless you have a very good system of points, then you will probably get an error B, integer out of range error, with a large interpolation, so be careful!

Input your start and stop values for the X axis. This is the area to pay attention to if you are interested in forecasting and trend prediction. Leave enough "headroom" and specify the screen limits, and your ZX computer will plot your trends. Input the desired point density (the number of points to be plotted on the screen, 40 is pretty good) and let it run for about 20 seconds.

4. You will be back at the plot menu now. This time, when you plot your graph, specify curve fit and see your results. The curve statistics will be lprinted if you press "Z."

If you rescale your graph, then rescale your fit.

Choice four will bring you to the print menu. If you desire lprinting, then add an lprint line where it is required. You may print from your data (and correct it in the direct mode), your curve or any other curve you can think of. When using interpolation, if you specify all of your data, and the degree of fit versus the number of data points DON'T match, then you will be asked if you would like a listing as plotting multiple interpolations is not implemented. More programming is required to implement this capability.

If you go back to the main menu (choice zero), then you can go out to the integration subroutine. Pick your choice of routines and set your integration limits. If you do not want to input a new function, then enter zero (0) when asked. When finished, you will go back to the main menu. If you care to plot the same points again, then STOP (0) and GOTO 666. This will bring back the PLOT menu.

Enjoy it!

'SE' in Action!

To give you an idea of the power of SE, I have listed some of the menus available. I input a short table of data, plotted it and fit it with several different curves. You can rescale and enlarge an area if you want. My enlargement is fit with a cubic interpolating polynomial. Boy, what a good fit!

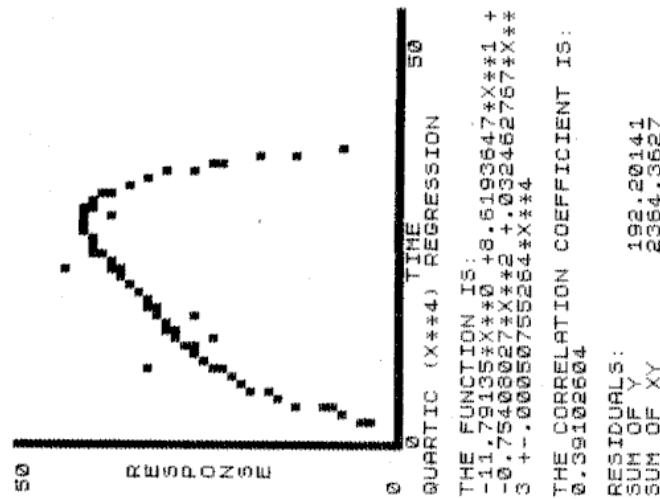
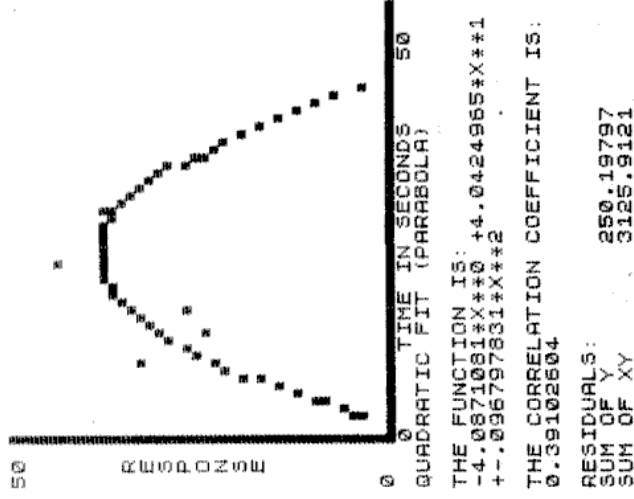
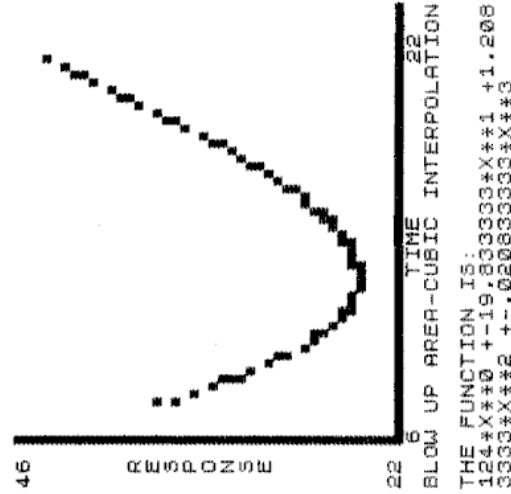
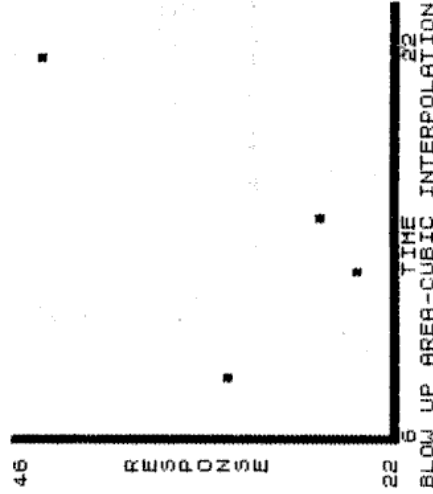
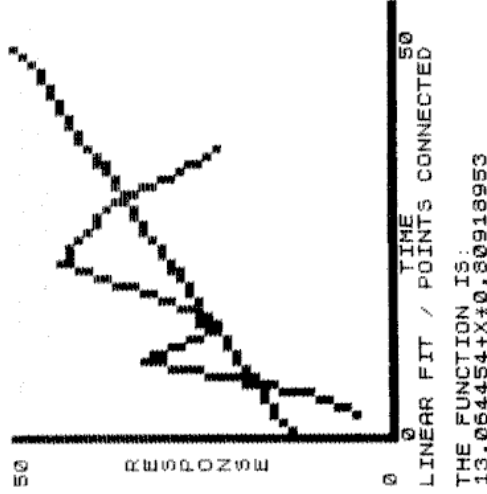
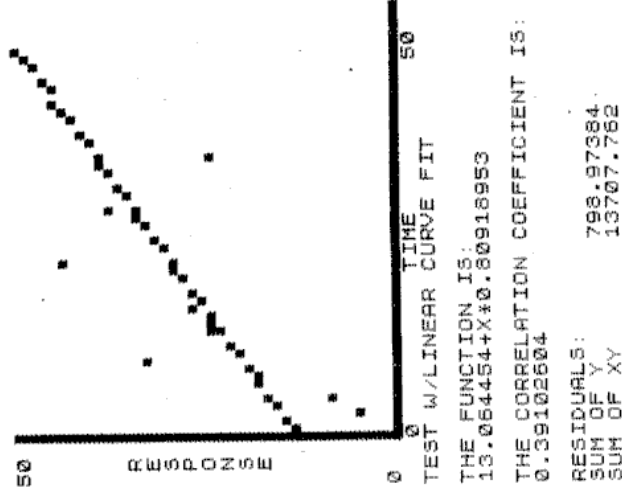
The data for each graph is identical. Only the curve fit has been changed in each case.

SAMPLE DATA POINTS		
PAIR	X-VALUE	Y-VALUE
1.	2	4
2.	4	8
3.	8	32
4.	12	24
5.	14	26
6.	20	44
7.	26	38
8.	32	24

---SAMPLED MENU---
BY THOMAS BENT

OPTIONS:

- 0) STOP
- 1) ADD 2 MATRICES
- 2) SUBTRACT 2 MATRICES
- 3) MULTIPLY 2 MATRICES
- 4) EVALUATE A DETERMINANT
- 5) SIMULTANEOUS LIN. EQUATIONS
- 6) PRINT / PLOT AND FIT DATA
- 7) INTEGRATION



MATH DEVELOPMENT Part 4

With the continuing upheaval in the computer industry, one can only speculate half-heartedly about the future of computers. They won't go away, but how many of the different types will survive? How advanced will they get and how much will they cost? Whatever those answers may be, the reasons for using computers remain the same. Two of the current philosophies are string manipulation (word processing, communications) and number manipulation (spreadsheets, games). The computer thrashes blindly through memory locations doing its' job. The difference between two programs lies in the sequence of execution or algorithm, if you will. The crux here is, how do you write the perfect algorithm for your given job? First, you need a job to do. You then must realize that a computer is just a tool; it only does what you tell it and finally; practice, practice, practice!!

Knowing how to program does not get you very far any more. My five year old son can do very basic programming. You need more, much more. A solid math background is a good start regardless of other expertise. A good graphic game that has more than straight line movement requires some premeditated algorithm to accomplish that move. Once the computation is made, the move can be stored and regenerated instantly with minimum effort. If you don't know the algorithm then forget it. Your game's a bore. Computer simulations are pre-planned, they don't just happen. People slave for months (right Fred?) over a great program.

Dissemination of information is what this magazine is all about. Many computer magazines support nothing but product reviews-all bun, no beef! In my series on math, some of the routines were translated directly from Fortran (Gauss Elim.) while others were original (Cramers rule any size, interpolation) along with plenty of space saving tips. SE in its' glory is a 20K ASCII file in 12.5K Sinclair basic (12 page listing). There is still plenty to be done. Still, no matter how many subroutines you have at your disposal, you need some knowledge of how to use them. My philosophy all along has been to explain the basics of the algorithms in the programs. They were all hand derived at some point in the past. The last major math installment follows; Polynomial and real time differentiation (derivative).

Differentiation

This is a vastly important subject. Your mind is extremely fast at calculating a derivative. Every time you catch a ball or stop for a traffic light, your mind is busy differentiating, among other things. Differentiation is the opposite of Integration. An integral covers an area or is total work as noted by addition (miles, minutes, your electric meter reading) or multiplication (eg. sq. ft., ft. lbs., mile-minutes, volt-amps, kilowatt-hours) where differentiation is denoted by division (eg., velocity or speed in ft/sec, or force in lbs./inch). To make a distinction, acceleration is a second derivative (ft/sec/sec), or the derivative of

the first derivative (velocity). In other words, speed changes with time. Totally confused yet? Sorry, just note that derivatives are a small piece of what is happening to the whole function at a given moment.

When you have a lot of data, it is easy to plot and see what is going on with the trends and tendencies of your information. If you have precious few bits of information, then it is very helpful to know a lot about those few points. The derivative is the means by which you develop information about your points. In simple terms, on an X,Y coordinate system, the derivative is the rise over the run (Y/X). It is noted as dy/dx or $\Delta Y/\Delta X$. These two terms have different meanings, but are identical for linear models. These Δ and dx functions stand specifically for the change in Y with respect to X. X will usually be time for real time data, but it can be anything we want it to be. Time is nice since it is impossible to sample two data points and have no dx (change in time).

Should you get out your old (or new) Calculus book, you can look up the formula for calculating a derivative. It looks something like this:

$$f'(X) = ((f(X+dx) - f(X))/dx)$$

You may find this a little easier to see:

$$dy = (Y(I) - Y(I-1)) / (X(I) - X(I-1))$$

If you are familiar with trigonometry, then you may recall that Y/X or dy/dx will give the tangent of the angle between the X axis and a straight line denoting the change in Y values. This makes for easy plotting of data by summing values (Logo like). Since Y values are plotted directly above the X values then,

$$Y = Y(I-1) + (X(I) - X(I-1)) * \tan((Y(I-1) / (X(I) - X(I-1)))$$

Now, you may think that this is just a bit cumbersome to calculate for each value you read (that's INPUT in Sinclairese) in, but after the first time through it becomes much easier. Y becomes $Y(I-1)$ and $X(I) - X(I-1)$ can be dx . The above formula simplifies to:

$$Y = Y + dx * \tan(Y/dx)$$

You then need only PLOT X,Y (provided they are on scale). This method is still somewhat inconvenient and all it really does (in this form) is guess at what the new Y value is. The next Y value to be read would be the actual Y value. If these values are not equal, then you are not dealing with a linear function. This does shed some light on whether your function is increasing at a faster rate or a slower rate. This is in fact some indication of the second derivative. An example would be a time, distance vector. If time values are plotted on the X axis and Y values are the distance you travel after a certain time, then it is easy to see that at a constant speed (a linear-straight line function) you can predict very easily where you will be at any time. Speed is the first derivative. At 60 (MPH), you will move 88 feet in the next second. Air Force

bombardiers thrive on first derivatives. However, if you have a lead foot like myself and accelerate until it is necessary to brake to avoid hitting the car in front, then you can not accurately predict distances versus time. Bombardiers aren't thrilled about second derivatives.

One more thing to confuse you about these functions is the sign of each (first and second) type of derivative is independent of the other. If both derivatives are positive, then the Y values are getting larger faster. If the first is positive and the second is negative, then the Y values are getting larger slower. If the first is positive and the second is zero, then all Y values in that region should lie on a straight line (increasing distance-slope up). Similarly, if both are negative, then you are losing ground faster and faster. With the second derivative positive, you are slowing down (but still going backwards). Again, if the second derivative is zero while the first is negative, the speed is constant (linear, backwards or decreasing-slope down). If the first derivative is zero, this implies the the distance is constant or, you have stopped.

Nothing to it, right? Any good math book will say that that is just how derivatives operate. The engineer will say that it sure looks good on paper. Don't program your robot to pour a cup of coffee this way though. The problem remains that the source of the data is subject to spikes and transmission errors. Even though they may be one in a billion, bad data can make that coffee fall right in your lap! Taking two points to evaluate a derivative implies a mid-point value and not a true real time value. The solution to high speed related rates type problems lies in not only software, but hardware as well. Fast data acquisition may be vital. However, if your robot is not running a horse race, then software can do the job. Taking two data points very close together may get you closer to a real time derivative but it will not improve your accuracy too much and you are still subject too data error. Taking more data points will help much more although your true evaluation point may be slightly further from a present time position.

Why worry at all about the derivative? What practical use is it? The problem of starting, moving and stopping a robot smoothly has plagued AI (artificial intelligence) scientists for years. There are many sophisticated schemes used to manipulate robots. Still, the manual 'LEARN' mode prevails at this time. Predetermined algorithms are being developed right now. The best AI books available are just collections of papers by various experts. (This is still a wide open field for anyone to partake). Getting back to basics, data acquisition and display can be handled conveniently with present day knowledge. Two easy ways to keep track of relative changes in your data are moving trend line (eg., stock market) and polynomial differentiation. There are two methods of differentiating polynomials as well. One is with a few points and the other is with a function (of course). If data input is sporadic, then a trend line is advantageous. On the other hand, regular data input (without lots of spikes and noise) can take advantage of an N-Point (differentiating an interpolating polynomial) algorithm. If you desire to study the data by collecting and 'Post-Processing', then fitting and differentiating the function will do nicely.

The first table is composed of the N-Point functions. The more points, the more accuracy. Three and five points are particularly nice because the mid-point (which happens to be the point you are evaluating the derivative for) drops out and leaves one less calculation. Besides, 4 point actually require 7 values although three drop out of the equation. The convention used in the following table has (I) as the point for which the derivative is being calculated for. Remember that the values are interpolated. Evaluating trigonometric functions like sines and cosines requires more than 4 points. In SWN 1/3, the error for interpolating holds for differentiating also. For N=4 and N=5 points, there are third derivatives and N=5 has a fourth derivative. If you have interest in these, send us a SASE and we'll send them to you.

N-POINT DIFFERENTIATION IN THE MIDDLE OF AN INTERVAL

N=3	$dy = (-Y(I-1) + Y(I+1)) / (2 * dx)$ $d2y = (Y(I-1) - (2 * Y(I)) + Y(I+1)) / (dx ** 2)$
N=4	$dy = (Y(I-3) - 27 * Y(I-1) + 27 * Y(I+1) - Y(I+3)) / (48 * dx)$ $d2y = (Y(I-3) - Y(I-1) - Y(I+1) + Y(I+3)) / (2 * dx ** 2)$
N=5	$dy = (Y(I-2) - 8 * Y(I-1) + 8 * Y(I+1) - Y(I+2)) / (12 * dx)$ $d2y = (-Y(I-2) + 16 * Y(I-1) - 30 * Y(I) + 16 * Y(I+1) - Y(I+2)) / (12 * dx ** 2)$

As in past issues, I always like to give more than one way to accomplish a task if possible. Different circumstances require a change of tactics. Interpolating, means estimating an exact course between known values. These estimations can (but not necessarily) gyrate strangely. As in least squares fitting (SWN 1/3), if you are interested in the trend of the change over a length of time or distance, a simple least squares fit will do. When I say simple, I mean just that. If data is coming in at set intervals, then dx (as well as all X) is a constant. Once the interval is defined, over half of the least square calculations drop out. It becomes necessary to keep track of only Y values. If you only want to keep track of a given amount of data and can discard the rest, then a moving trend line will give a very stable derivative (the stock market should be so lucky). To accomplish this, just divide or subtract out the old Y value and multiply or add in the new one. Recalculate the slope (M), the intercept (B) will become a pivot (constant), and you have an indication of the first derivative. The values will be relative but will show the trend nicely. If you save the old M, then subtracting the old M from the new M will give you an indication of the second derivative. Simple and fast! One hitch, after subtracting, dividing, etc., on the same set of numbers for long periods of time, watch out for roundoff error. Every so often you should reinitialize your array and start fresh again. If your X values are at varying times, then you must redo the whole least squares after each point. Cramer's Rule is pretty quick though. The listing given runs in about 0.5 seconds in fast, 3.6 in slow. That includes those time consuming extra scrolls and prints also. This technique is certainly not instantaneous, but spikes in the incoming data will be averaged out quite nicely. If the data rate is fairly slow, then higher orders of derivation (a bigger set of simultaneous equations - remember them?) can be employed to get a more accurate derivative. Least squares routines utilize the partial derivatives of each degree of a function to evaluate a fit. Any degree of fit can be achieved if wanted or needed. This method only gives the trend through the entire fit area. However, if the function is developed, then the next little technique will give the instantaneous derivative for any spot inbetween the first and last points defined.

Horner's Rule

To ease the implementation of this technique, we must set up our function so as to evaluate it with the utmost speed and efficiency. The technique is called Horner's Rule (HR). This simple method reduces polynomial calculations by a third and gets rid of those slow exponent evaluations. We can set up a loop or fill a string with a function. The loop method is more dynamic, but the string method will retain the function for whatever else you can cook up.

Use Horner's Rule wherever it fits; it is just good programming.

This very short routine takes the coefficient (X(I)) of the highest power times the value of the point you are evaluating, adds it to the next lower

power X(I) and loops back to do it again. This continues for all terms.

The Horner's Rule conversion looks like this:

$$4*X**3+2*X**2-3*X+1 \text{ becomes the following,}$$

$$((4*X+2)*X-3)*X+1$$

The first time through the loop in line 9550, will give the Y value or is the same as using VAL F\$. I believe that this routine is faster than VAL since there are no powers, but I'm not sure. Besides, the following times through the loop gives all of the derivatives. The swap terms routine is necessary to get the powers lined up in the right order to take advantage of the routine. If you use the interpolation routine, then you can bypass the swap routine completely. The create function routine is just to compare the Val function with the HR routine. The highest power coefficient should be entered as X(1) and so on, and they must be present before you 'create' your function. A function in F\$ is not necessary to evaluate the derivative though. Just put X in an incrementing loop and GOSUB listing 3. you can watch all the derivatives change with each point evaluated.

I certainly hope you have enjoyed and comprehended this Math series. I know how difficult this material is to learn. Believe me, it takes a lot of patience and PRACTICE!

Differentiation Demo

```

10 REM SINE TREND
20 SAVE "MATHS"
30 REM LISTING NO. 2
40 LET N=4
50 DIM Y(N)
60 LET M=0
70 SLOW
90 REM TIME INTERVAL
100 LET T=3
110 LET T1=0
120 LET T2=0
130 LET Y=0
140 LET TY=0
150 LET X=0
160 FOR I=1 TO N
170 LET T1=T1+T
180 LET T2=T2+T*T
190 LET Y(I)=0
200 LET Y=Y+Y(I)
210 LET TY=TY+T*Y(I)
220 LET T=T+T
230 NEXT I
240 LET CR=N*T2-T1*T1
250 IF NOT CR THEN STOP
260 REM ROUTINE STARTS HERE
270 FOR J=1 TO N
280 LET M1=M
290 LET M=(N*TY-Y*T1)/CR
300 LET M2=M-M1
310 LET Y=Y-Y(J)
320 LET TY=TY-T*Y(J)
330 LET X=X+3
340 SCROLL
350 SCROLL
360 REM PRINT OR PLOT Y,M AND M
2
370 PRINT Y(J),M
380 SCROLL
390 SCROLL
400 PRINT ,M2
410 REM LET Y(J)=USR INPUT (Y(J))
420 LET Y(J)=SIN (X/180*PI)
430 LET Y=Y+Y(J)
440 LET TY=TY+T*Y(J)
450 REM PEEK FRAMES IN SLOW TO
    GET LOOP TIME (SECONDS)

```

```

460 REM PRINT (PEEK 16436+256+R
EEK 16437)/-50+1092.25
470 REM MULTIPLY BY 6.75 FOR FA
ST MODE
480 REM PAUSE DELAY TIME OR GOS
UB OTHER ROUTINE
490 NEXT J
500 GOTO 270
510 REM
520 REM LISTING NO. 1
530 REM REAL TIME DIFFERENTIATI
ON
540 REM LET DX=1
550 LET DX=30*PI/180
560 SLOW
570 FOR I=0 TO 90 STEP 10
580 LET X=I*PI/180
590 LET DY1=(SIN (X-2*DX)-8*SIN
(X-DX)+8*SIN (X+DX)-SIN (X+2*DX
))/(12*DX)
600 LET DY2=(-SIN (X-2*DX)+15*S
IN (X-DX)-30*SIN X+15*SIN (X+DX)
-SIN (X+2*DX))/(12*DX*DX)
610 LET ER1=DY1-COS X
620 LET ER2=DY2-(-SIN X)
630 SCROLL
640 PRINT "DEG 1ST. DER.", " ER
ROR"
650 SCROLL
660 PRINT I; " ";DY1
670 SCROLL
680 PRINT ,ER1
690 SCROLL
700 PRINT "2ND. DER.", " ERROR"
710 SCROLL
720 PRINT DY2,ER2
730 SCROLL
740 NEXT I
750 STOP
760 REM
4740 REM LISTING NO. 3
4745 REM FROM SE
4749 REM (1 DIM )SWAP TERMS
4750 LET L=M
4760 FOR I=A TO M/2

```

```

4770 LET T=X(I)
4780 LET X(I)=X(L)
4790 LET X(L)=T
4800 LET L=L-1
4810 NEXT I
4814 REM INSERT NEXT LINE IN SE
4815 IF Z=7 THEN RETURN
4819 REM CREATE A POLYNOMIAL FUN
CTION (FROM SE)
4820 LET F$=""
4830 FOR I=A TO M
4850 LET F$=F$+STR$ X(I)+"*X**"+
STR$ (I-1)+" "
4860 IF I<>M THEN LET F$=F$+"+"
4870 NEXT I
4879 REM CHANGE 4880 TO :
4880 IF NOT AL THEN GOTO (5486 A
ND Z<7)+(9260 AND Z=7)
8998 REM
8999 REM
9000 REM POLYNOMIAL DERIVATIVE
9100 REM INPUT F$ OR FIT WITH SE
PROGRAM
9110 REM INPUT COEFFICIENTS AS X
(J) OR GET FROM SE
9230 LET Z=7
9240 LET AL=NOT PI
9250 REM GOTO 4820
9260 REM HORNER'S RULE IN A LOOP
9265 PRINT F$
9270 DIM B(M)
9280 GOSUB 4750
9350 REM DERIVATIVES
9500 FOR I=1 TO M
9510 LET B(I)=X(I)
9520 NEXT I
9530 FOR J=1 TO M
9540 FOR I=2 TO M+1-J
9550 LET B(I)=B(I)+B(I-1)*X
9560 NEXT I
9570 PRINT "DER. ";J-1;" AT X =
";X;" = ";B(I-1)
9580 NEXT J
9590 GOSUB 4750
9600 RETURN

```

Intro to Integration

If you had trouble following Tom's integration article, here's a presentation of computer integration from more of an applications standpoint. After you've gotten your toes wet with this article, you might find that you can now dig out those dusty textbooks and, with Tom's help, integrate anything any old way.

Though Tom refers to the core routine in this program as "Fred Rule," I have to cop to the fact that I originally got it from some unsung (and unlisted) programmer involved with writing the TRS-80 Pocket Computer manual. Who knows where it came from originally - probably some fellow in the 60's on a discrete-transistors mainframe.

INTRO TO INTEGRATION

RAM REQUIRED - 1K up.

This program performs the calculus operation of integration, which is useful for finding areas under curves, computing average and root-mean-squared (RMS) values, and anytime you need to know how much of an accumulation of something occurs as a function of time or some other variable. For example, if a graph shows daily gross income, then the integral of the graph over a certain time span would represent total gross income received during that time. From this you can compute average daily income by dividing by the number of days in the time span you added up, or integrated. In electronics it is useful to find average levels of various voltage or current waveforms.

INTEGRATION fits into 1K, since the function values are calculated instead of stored in memory. As a result, it will only work with functions that can be expressed as an algebraic expression.

```

2 FAST
5 PRINT TAB 10;"INTEGRATION"
7 PRINT "INPUT FUNCTION F(X)"
"Y="
8 INPUT B$
9 CLS
10 PRINT "Y=";B$;TAB 0;"X(0)="
20 INPUT X0
30 PRINT X0
40 PRINT "X(F) =";
50 INPUT XF
60 PRINT XF
70 PRINT "NO. DIVS=";
80 INPUT D
90 PRINT D
100 LET B=(XF-X0)/2/D
110 LET A=0
120 LET X=X0
130 LET Y=VAL B$
140 LET A=Y+A
150 LET X=X+B
160 LET Y=VAL B$
170 LET A=Y+A
180 LET X=X+B
190 LET Y=VAL B$
200 LET A=Y+A
220 IF XF-X>1E-6 THEN GOTO 140
230 LET S=A*B/3
240 PRINT "INTEGRAL=";S
250 PRINT "AVERAGE=";S/(XF-X0)
260 PRINT "SAME F(X)?"
265 PAUSE 40000
280 IF INKEY$="Y" THEN GOTO 9
290 IF INKEY$="N" THEN RUN
295 CLS

```

Enter RUN to start the program. You are asked to INPUT an expression of the function to be integrated, in terms of independent variable X. For example, a sine wave with a peak amplitude of 100 volts can be represented by $F(X) = 100 \cdot \sin X$. Entering this, the computer prints the function and prompts the limits of integration, i.e. from what starting point X_0 to what ending point X_F . If we want the average level or integral over a full cycle of the wave you would enter zero and 2π , respectively. You then enter how many divisions you want the function broken into; generally, higher numbers give better accuracy but this can be offset by inaccuracies in evaluating the function, etc. - the recommended range for typical functions is 10 - 50. Entering 10, the computer proceeds to sum the integral, printing it and the average level. In this example, the answer is very close to zero, as expected since the "negative area" of the second half of the wave exactly cancels the "positive area" of the first half.

To use the same function in another problem, press "Y". Let's try it, and this time make $X_F = \pi$ - i.e., we're only integrating over the first half-cycle. This time the value of the integral is 200.00068, very close to the exact value for this problem of 200. Also as expected, the average voltage over the half-cycle is 63.622, or about 63.7% of the peak value. If you ever wondered how this value was derived, now you know.

The program uses Simpson's Rule to evaluate the integral, the most accurate approximation for many functions (accurate up to third degree polynomial). If you're curious, look up Simpson's Rule and work out how the program performs the operation.

RMS CALCULATOR

You can modify the INTEGRATION program to calculate RMS instead of average values. The RMS value is also called the "effective" value, as it represents the DC voltage or current that would produce the same average power in a fixed resistance (linear circuit). Delete line 240, and change/add the following lines:

```
5 PRINT TAB 15;"rms"
135 LET Y=Y*Y
165 LET Y=Y*Y
195 LET Y=Y*Y
230 LET S=SQR ((A*B/3)/(XF-X0))
250 PRINT "rms value=";S
```

Use the $Y = 100 \cdot \sin X$ example as above, integrating first over a half-cycle ($X_0 = 0$, $X_F = \pi$) and then over a whole cycle ($X_0 = 0$, $X_F = 2\pi$). The answers are the same this time. (Why?)

You can use this to find the RMS value of any function that can be algebraically expressed (and therefore can be evaluated by computer). In this program as well as the INTEGRATION program, it is alright to enter an odd number for "No. of Divs." This is because the function is actually evaluated at twice the number of data points as specified by "No. of divs." so the total number of points is always even (a requirement for correct results using Simpson's Rule).

Arbitrary Functions

Not all real functions can be conveniently expressed as an algebraic formula, of course. Sometimes we only have values from a graph or table that we have to integrate. In this case, we can store a function values in an array; with a 2K machine it is not unreasonable to store a function with 100 data points. The following program will integrate an arbitrary function, with resolution limited only by available memory space. After the program is a sample run illustrating a possible application. You can SAVE the program and data to tape by pressing "S" when output appears.

2 SLOW	DIV X=	Y=
5 PRINT TAB 11;"INTEGRATION";TAB	0 0	0
B 10;"INTEGRATION"	1 0.5	0
10 PRINT "NO. OF DIVS (MUST	2 1	0
BE EVEN) =";	3 1.5	0
15 INPUT D	4 2	0
20 IF INT (D/2) <> D/2 THEN GOTO	5 2.5	0
15	6 3	0
25 PRINT D	7 3.5	0
30 DIM A(D+1)	8 4	0.5
40 PRINT "X(0) =";	9 4.5	1.7
50 INPUT X0	10 5	2.9
60 PRINT X0	11 5.5	4.8
70 PRINT "X(F) =";	12 6	7.35
80 INPUT XF	13 6.5	10.2
90 PRINT XF	14 7	12.5
100 PRINT "INPUT FUNCTION VAL	15 7.5	14
UES"; AT 20,0;"DIV X=";"Y="	16 8	17.3
110 FOR B=0 TO D	17 8.5	18.3
115 LET X=X0+(XF-X0)*B/D	18 9	18.8
120 SCROLL	19 9.5	19.5
130 PRINT AT 21,31;" ";AT 21,0;	20 10	20.3
B,TAB 4;X;AT 21,16;"?"	21 10.5	22.6
140 INPUT A(B+1)	22 11	25.8
145 PRINT AT 21,16;A(B+1)	23 11.5	29.5
150 NEXT B	24 12	32.3
160 FAST	25 12.5	35.5
170 PAUSE 40000	26 13	35.7
200 LET C=0	27 13.5	34.2
210 LET A=0	28 14	31.3
220 LET Y=A(1)	29 14.5	28.8
230 LET A=Y+A	30 15	27.7
240 LET C=C+1	31 15.5	28.2
250 LET Y=A(C+1)	32 16	27.8
260 LET A=Y*4+A	33 16.5	24.3
270 LET C=C+1	34 17	21.5
280 LET Y=A(C+1)	35 17.5	18.3
290 LET A=Y+A	36 18	12.2
300 IF C<D THEN GOTO 230	37 18.5	7.5
310 LET S=A*(XF-X0)/D/3	38 19	4.2
320 CLS	39 19.5	2.1
330 PRINT "INTEGRATION=";S	40 20	0.4
340 PRINT "INTEGRATION=";S/	41 20.5	0
(XF-X0)	42 21	0
350 PRINT "SAVE - "S""	43 21.5	0
360 PAUSE 40000	44 22	0
370 IF INKEY\$="S" THEN SAVE "AR	45 22.5	0
BIN"	46 23	0
	47 23.5	0
	48 24	0

SAMPLE RUN: Let's assume we're interested in finding the total energy produced by a solar panel over the course of a day. We can measure the panel's output, in watts, at regular intervals (e.g. every half-hour) to arrive at the table shown. The results show that this solar panel put out a total of 299.18 watt-hours on this particular day, with an average output of 12.466 watts over the 24-hour period.